VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Inteligentní dopravní systémy
Intelligent Traffic Systems

2011 Andrej Matejčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Andrej Matejčík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Inteligentní dopravní systémy
Intelligent Traffic Systems

Zásady pro vypracování:

Cílem práce je vytvoření modulů rozsáhlejšího systému pro podporu analýzy a simulací v oblasti dopravy. Výsledný systém je výstupem několika diplomových prací, kdy tato práce pokrývá především oblast architektury webových služeb a realizaci GUI.

1. Zanalyzujte a popište možnosti a přístupy realizace komunikační sběrnice na bázi SOA pomocí technologií MS .NET WCF.
2. Navrhněte takovéto rozhraní na bázi webových služeb, které bude schopno řídit komunikaci a vazby mezi službami, dynamicky upravovat portfolio služeb a realizovat hlavní vrstvu výměny dat a služeb v systému.
3. Implementujte klíčové prvky této architektury.
4. Zanalyzujte a popište možnosti tvorby GUI v MS Silverlight, a to s ohledem na požadavky aplikace pracující s mapovým podkladem a napojení na výstupy analýz a simulací.
5. Implementujte GUI v MS Silverlight s ohledem na výše vyvinutou architekturu služeb.
6. Zhodnoťte navržené řešení s ohledem na možnosti reálného nasazení.

Seznam doporučené odborné literatury:

J. Lowy: Programming WCF Services, 2007, O'Reilly Media, ISBN: 978-0596526993
M. MacDonald: Pro Silverlight 4 in C#, 2010, Apress, ISBN: 9781430229797

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Radecký**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

....

Dátum Podpis autora

Abstrakt

Témou písomnej správy je rozbor problematiky tvorby agilného a konkurencieschopného systému v oblasti dopravy. Okrajovo vykresľuje význam IDS v reálnom svete. Správa popisuje možnosť začlenenia vyvíjanej aplikácie do iných systémov za účelom zvýšenia jej použiteľnosti. V snahe poskytnúť čitateľovi popis nosných technológií a techník využívaných pri budovaní výsledného produktu, popisuje dôvod ich použitia, ktorý sa vzťahuje buď k určitej požiadavke IDS, alebo problému. V snahe vytvoriť kvalitnú architektúru softvérového diela, správa popisuje dostupné možnosti SOA na platforme .NET. Navrhuje implementáciu kľúčových prvkov aplikácie. Taktiež odkrýva spôsob interakcie medzi klientom a serverom. Približuje spôsob vývoja klienta pomocou platformy Silverlight 4.

Kľúčové slová

Systém, architektúra, SOA, Silverlight, doprava, WCF RIA Services, WCF, webová služba, interoperabilita, modul, vrstva

Abstract

The subject matter of the master´s thesis is the analysis of an agile and competitive system creation in the field of transportation. The thesis marginally describes the significance of ITS in practice and the possibility to involve the application under development in other systems to enhance its applicability. To provide the reader with the description of key technologies and techniques applied to develop the final product the thesis describes the reason of their application that relates to a particular ITS demand or a problem. In order to create a perfect architecture of the software product, the thesis offers available SOA solutions at the NET platform. It suggests implementation of key elements of the application and reveals the way of interaction between the client and the server. The method of client development by using the Silverlight 4 platform is illustrated.

Key words

System, architecture, SOA, Silverlight, traffic, WCF RIA Services, WCF, web service, interoperability, module, layer

Zoznam použitých symbolov a skratiek

CLR – Common Language Runtime

DB – databáza

EF – Entity Framework

HTTP – Hypertext Transfer Protocol

IDS - Inteligentný dopravný systém

ITS – Intelligent Traffic System

JAX-WS – Java API pre XML webové služby

JSON – Javascript Object Notation

MIME - Multipurpose Internet Mail Extensions

MSMQ – Microsoft Message Queuing

MTOM – Message Transmission Optimization Mechanism

REST – Representational State Transfer Protocol

RIA – Rich Internet Application

RPC – vzdialené volanie procedúr

SOA – servisne orientovaná architektúra

SOAP – Simple Object Access Protocol (Service Oriented Architecture Protocol)

SVN – systém pre správu a verzovanie zdrojových kódov

TCP – Transmission Control Protocol

UI – užívateľské rozhranie

URL – Uniform Resource Locator

WCF – Windows Communication Foundation

WSE – Web Services Enhancements

XML – Extensible Markup Language

Obsah

1 Úvod 8
1.1 Cieľ práce 9
1.2 Popis práce 9
1.3 Inteligentné dopravné systémy 9
1.4 Systém Floreon+Traffic 10
2 Požiadavky na vyvíjaný systém 11
3 Budovanie systému pomocou SOA modelu 12
3.1 Technológie pre SOA v prostredí .NET 14
3.1.1 Čo je WCF 14
3.1.2 Web Services (ASMX) 16
3.1.3 WCF (core) 17
3.1.4 WCF Data Services 19
3.1.5 WCF RIA Services 21
3.2 Výber aplikačného modelu pre budovanie SOA 22
4 Popis architektúry navrhovaného systému 22
4.1 Štruktúra jadra aplikácie 23
4.2 Rozšírenie architektúry o nový modul 25
4.3 IDS ako modul väčšieho systému 26
5 Realizácia požadovanej funkcionality 28
5.1 Prístup k dátovému zdroju a spolupráca s DB serverom 28
5.1.1 Primárny spôsob komunikácie s databázou 29
5.1.2 Práca s priestorovými dátami v databáze 30
5.2 Budovanie klienta IDS 32
5.2.1 Komunikácia pomocou asynchrónneho programovacieho modelu 33
5.2.2 Bezpečnosť komunikácie medzi klientom a serverom 36
5.2.3 Klient v spolupráci s mapovou komponentou 39
5.2.4 Zobrazenie štatistických výsledkov na klientovi 41
5.3 Popis ďalších komponent systému 41
6 Hodnotenie vytváraného systému 42
7 Záver 43
8 Literatúra 45

Prílohy

A Manuál pre tvorbu webových služieb 47
A.1 Ako tvoriť ASMX webové služby.....47
A.2 Ako tvoriť webové služby pomocou klasického WCF.....49
A.3 Ako tvoriť webové služby pomocou WCF Data Services.....50
A.4 Ako tvoriť webové služby pomocou WCF RIA Services 51
B Tvorba dátového modelu pomocou Entity Framework.....55

1 Úvod

Mobilita je dôležitým faktorom z hľadiska kvality života každého jednotlivca. Spôsob, akým sme schopní sa premiestňovať z jedného miesta na druhé, získava čoraz významnejšiu váhu z pohľadu skvalitnenia životnej úrovne. Doprava je neodmysliteľnou súčasťou našej spoločnosti. Súčasnosť nám umožňuje rôzne alternatívy prepravy, ako cestná doprava, letecká doprava, námorná doprava, klasická chôdza a mnoho ďalších. Čas je aspekt, ktorý by chcel každý čo najlepšie využiť. Viacerí z nás by privítali dlhšie dni, ktoré by pravdepodobne vniesli do našich myslí viac pohody a pozitívnej energie. Dopravné zápchy, nehody, nevhodné podmienky na trasách sú negatívne faktori, ktoré dokážu naštrbiť psychiku nejedného z nás. Komfort a plynulosť sú predpoklady, ktoré by mali získať reálnu podobu v dopravnej infraštruktúre.

Obrázok 1: Dopravná situácia na cestách[1]

Moderná spoločnosť si je vedomá, že môže zefektívniť životný štýl. Jednou z alternatív ako je toto možné dosiahnuť, je využiť informačné technológie. Trend rozvoja IT má markantný vplyv na dopravný proces. Zdokonalenie premávky vedie k zlepšeniu nielen ekonomickej, ale aj sociálnej situácie. Narastajúci počet dopravných prostriedkov na cestách vytvára tlak na vývoj a progres v oblasti dopravy. Jej monitorovanie poskytuje reálnejší pohľad na situáciu na cestách.

[1] Zdroj : http://www.problogger.net/archives/2008/06/12/how-batch-processing-made-me-10-times-more-productive/

1.1 Cieľ práce

Z dôvodu neustále sa vyvíjajúcich informačných technológií je prirodzené ísť s dobou a snažiť sa poskytnúť čo najlepšie, resp. najaktuálnejšie riešenie v danej problematike. Ani IDS nie sú výnimkou. Evolúcia nám neustále poskytuje nové alternatívy pre ich vytváranie. Inovatívnosť je vlastnosť, ktorá je dnes zárukou kvality. V nasledujúcej práci sa snažíme ukázať možnosť budovania moderného informačného systému, ktorý spĺňa požiadavky vo svojom odbore. Cieľom je popísať využiteľnosť architektúry systému, ktorá umožňuje spolupracovať s viacerými alternatívami. Text by mal podčiarknuť hlavné body, ktoré vedú k tvorbe kvalitného IDS. Čitateľ by mal získať prehľad o možnostiach, ktorými to je možné dosiahnuť. Tieto alternatívy sa pokúsime priblížiť a popísať.

1.2 Popis práce

V závere prvej kapitoly predstavíme inteligentné dopravné systémy a ich zámer. Tiež v krátkosti popíšeme produkt Floreon+Trrafic, ktorý je podobného charakteru, ako nami budovaná aplikácia.

Druhá kapitola je venovaná požiadavkám na vyvíjanú aplikáciu a na vlastnosti, ktorými by mal výsledný produkt disponovať.

V tretej kapitole sa bližšie pozrieme na možnosti budovania servisne orientovanej architektúry. Postupne budeme analyzovať jednotlivé dostupné možnosti. Pokúsime sa poukázať na pozitívne resp. negatívne črty daných alternatív a nakoniec si vyberieme tú najvhodnejšiu pre náš produkt.

Postupným listovaním sa presunieme k textu, ktorý stanovuje architektúru IDS. Na túto problematiku sa zameria štvrtá kapitola. Popisuje otvorenosť architektúry aplikácie a jej schopnosť zapracovať sa do iného systému.

Piata kapitola poukazuje na využité techniky a správanie sa systému, kde riešenia sa budú odrážať od stanovenej štruktúry aplikácie, a tiež požadovanej funkcionality.

Šiesta kapitola vyhodnocuje systém z hľadiska nasadenia, resp. dostupnosti. Poukazuje na dobré programátorské spôsoby pre uľahčenie implementácie.

Posledná siedma kapitola prináša celkový pohľad riešiteľa na produkt. Prezentuje pozitívne a negatívne prvky, s ktorými sa študent stretol. Naznačuje aktuálny stav produktu a možnosti jeho ďalšieho vývoja v súvislosti s prácami podobného charakteru.

1.3 Inteligentné dopravné systémy

V súčasnosti sa vyvíjajú informačné systémy, ktoré sa venujú širokému spektru problémov. Aktuálny vývoj IT nezanedbal ani oblasť rozvoja softvéru, ktorý sa venuje dopravnému procesu. Informačné technológie zaoberajúce sa problematikou zlepšenia situácie v dopravnej infraštruktúre sa postupom času stávajú tými, ktoré nám dávajú zelenú na cestách.

Medzi hlavné črty a záujmy IDS patria:

- Zvýšenie plynulosti dopravných prostriedkov na trasách
- Zvýšenie efektivity dopravy

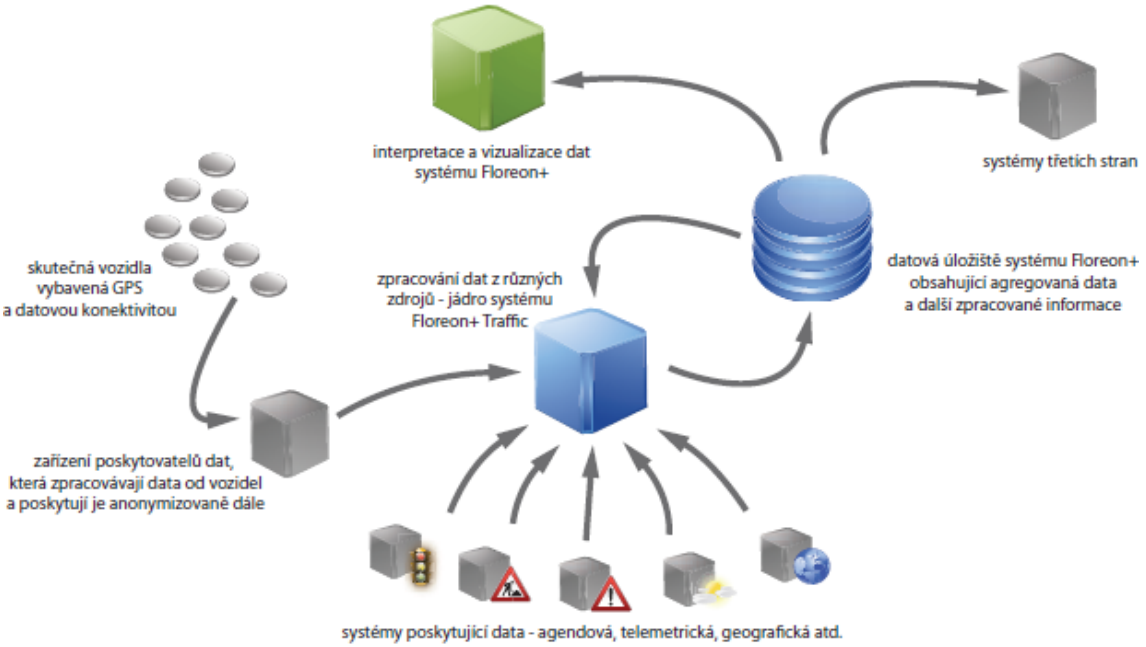
- Zníženie nákladov na prepravu
- Monitorovanie dopravy
- Bezpečnosť na cestách
- Zvýšenie komfortu cestujúcich
- Rýchly prehľad aktuálnej situácie na trase

1.4 Systém Floreon+Traffic

Medzi produkty zameriavajúce sa na situáciu na cestách, resp. mapovaním krízových situácií a ich riešením, sa stal systém Floreon+. Jedná sa o projekt vedeckovýskumného charakteru, ktorého hlavným cieľom je vybudovanie systému pre modelovanie, simuláciu a monitorovanie situácií spôsobených nielen prírodnými javmi, a to vrátane problematiky dopravy. Znalosť dopravnej situácie a možnosť využitia sofistikovaného aparátu, ktorý tieto informácie bude efektívne zhromažďovať a spracovávať, je zásadná pre celý rad činností nielen z pohľadu krízového riadenia, ale aj pre každodenný život.

Obrázok 2: Ukážka systému Floreon+Traffic

Základnou črtou Floreon+ je jeho modularita. Jedným z jeho komponent je systém Floren+Traffic. Princíp fungovania tohto modulu je založený na algoritmoch, ktoré sú schopné efektívne spracovávať veľké množstvo dát. Následne z nich získavať potrebné informácie a tie nakoniec vyhodnocovať s ohľadom na ich geografickú polohu. Vstupom sú dáta z rôznych zdrojov a v rôznych formátoch. Výstupom sú agregované informácie spojené s určitým miestom na mape. Pre lepšiu predstavu nám poslúži obrázok 3.



Obrázok 3: Princíp systému Floreon+Traffic[2]

Pohľad na zobrazenú architektúru indikuje efektívne získavanie prostriedkov, a to buď z komunikácií, alebo zo systémov, ktoré disponujú touto službou. Výsledné dáta sa ukladajú do dátového úložiska. Tieto výsledky sú ďalej zdrojom pre iné systémy, resp. sú priamo prezentované koncovému klientovi. Bližšie informácie o systéme Floreon+ nájdeme v zdroji [1], alebo na webovej stránke *floreon.vsb.cz*.

2 Požiadavky na vyvíjaný systém

Novovzniknutý systém by mal:

- byť schopný komunikovať s dátovým úložiskom
- efektívne pracovať nad získanými dátami z DB
- poskytovať klientom výstupy v správnom formáte
- zabezpečovať interoperabilitu s inými systémami

[2] Zdroj: [1]

- byť ľahko začleniteľný do väčších systémov (správna architektúra)
- poskytovať určitú úroveň bezpečnosti
- byť ľahko rozšíriteľný (o prípadný modul, resp. funkcionalitu)
- byť modulárny (pozostávať z viacerých modulov)
- efektívne komunikovať so Silverlight klientom (preferovaný klient)
- pracovať na klientovi s mapovým podkladom
- na klientovi poskytovať štatistické výsledky v prehľadnej podobe

Modularita je charakteristickým prvkom kvalitného systému, kde jeden modul môžeme zameniť za druhý bez toho, aby sme ovplyvnili zvyšné komponenty. Taktiež nám zabezpečuje zvýšenie úrovne jednoduchosti. Jednotlivé vrstvy zoskupujeme podľa ich zamerania a využitia. Pri tvorbe vrstiev je potrebné mať jasnú predstavu, akým spôsobom budú tieto časti medzi sebou komunikovať. Prepracovaná myšlienka jednotlivých vrstiev je cestou úspešnej architektúry. Cestou modularity systému sa v nasledujúcom texte vydáme aj my, kde si popíšeme funkcie a dôvod vzniku každého z kľúčových modulov.

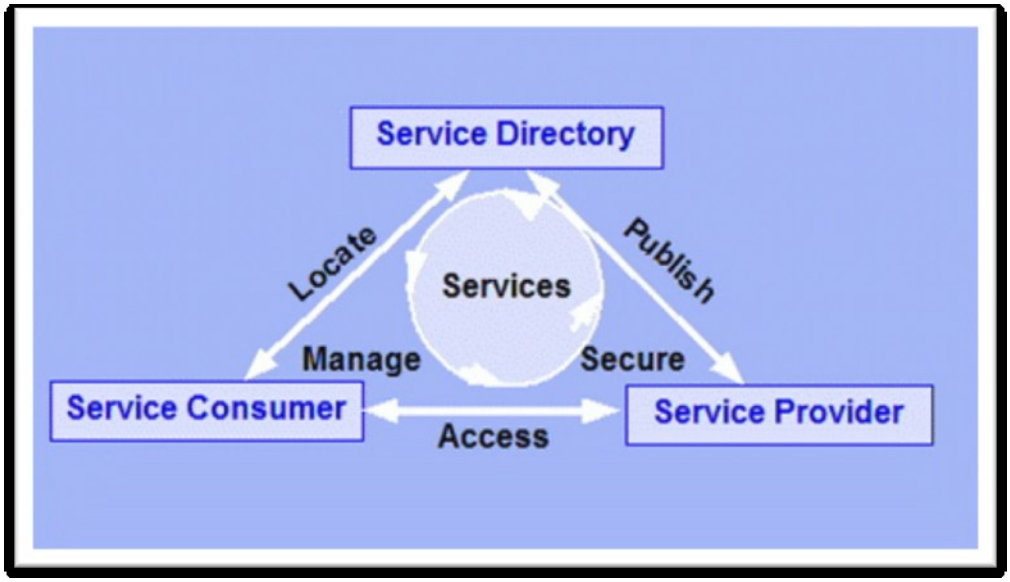
Architektúra je jedným zo základných faktorov, kde môžeme posúdiť kvalitu vyvíjaného produktu. Pozeráme sa na ňu a tvoríme ju na základe našich potrieb a požiadaviek. V prípade modifikovania charakteru systému by mala byť aplikácia z hľadiska architektúry škálovateľná. Z dôvodu budovania živého a perspektívneho systému, aplikovateľného aj v budúcnosti, si zvolíme cestu nazývanú SOA. Niektorí z nás si možno práve teraz položili otázku „Čo je SOA?''. V nasledujúcej kapitole si tieto nejasnosti vysvetlíme a pozrieme sa na možnosti, ktoré nám súčasný vývoj v IT predkladá.

3 Budovanie systému pomocou SOA modelu

SOA predstavuje množinu komponent, kde každá komponenta systému predstavuje službu. Služba znamená jednotku programu, ktorá reprezentuje určitý biznis proces. Pod termínom SOA sa neskrýva žiadna technológia, je to skôr návrhový vzor pre budovanie kvalitného, agilného a konkurencieschopného systému. Za týmito vlastnosťami sa skrýva model, pomocou ktorého zvyšujeme flexibilitu výsledného produktu s ohľadom na zmeny požiadaviek na daný systém.

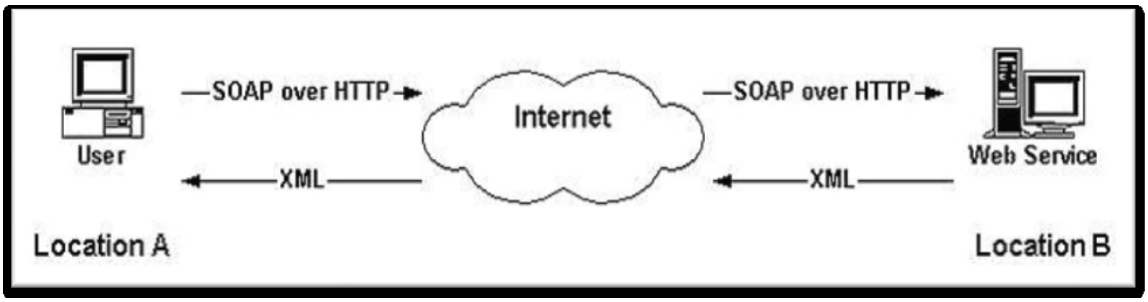
Prečo SOA:

- Interoperabilita
- Slabo-prepojiteľné väzby (loose coupling)
- Opakovaná použitelnosť (reuseability)
- Oddelenie implementácie služby od jej definície (rozhrania)



Obrázok 4: Základný princíp SOA z pohľadu ich publikovania a konzumovania[3]

Postupným vývojom Internetu vznikali nové štandardy, ako HTTP, HTML, resp. XML. Tieto pokroky zjednodušili vývoj protokolov webových služieb. Pri budovaní architektúry sa v súčasnosti môžeme obrátiť buď na SOAP protokol, alebo na REST model. Obidve tieto možnosti využívajú bežný HTTP protokol. Pri použití SOAP modelu sa zameriavame na biznis proces, ktorý je implementovaný webovou službou, ktorá tieto procesy publikuje navonok ako operácie. Na druhej strane, REST pracuje s dátami, resp. entitami, ktoré chceme publikovať v rámci služieb. URL adresa nám jednoznačne identifikuje zdroj, za ktorým sa skrývajú spomínané entity. REST sprístupňuje schémy, pomocou ktorých sú klienti schopní konzumovať dáta.



Obrázok 5: Schéma komunikácie cez SOAP protokol pomocou webových služieb[4]

[3] Zdroj: http://www.ibm.com/developerworks/webservices/tutorials/ws-soa-ibmcertified/section5.html
[4] Zdroj: http://www.c-sharpcorner.com/UploadFile/skulkarni/AuthenticateWebService11252005071231AM/AuthenticateWebService.aspx

Medzi možné alternatívy budovania SOA patria:

- JAX-WS – technológia v oblasti Javy, zaoberajúca sa budovaním webových služieb a klientov, ktoré používajú XML. Umožňuje vývojárom vytvárať webové služby, ktoré sú orientované na správy (*message-oriented*), resp. služby využívajúce RPC. Zabezpečujú interoperabilitu, a tým pádom služby tvorené v Jave sú dostupné aj pre odlišne zameraných klientov (napr. .NET klient). Platforma J2EE poskytuje viacero spôsobov, ako vytvoriť webové služby vyhovujúce našim požiadavkám. Dostupné informácie týkajúce sa tvorby webových služieb v Jave, je možné získať pomocou zdroja [2].
- MS .NET WCF – technológia fungujúca na .NET platforme. Taktiež zaručuje interoperabilitu a tvorbu služieb prezentujúcich sa navonok svojím rozhraním. Publikácia [3] prezentuje podrobné informácie ohľadom tvorby SOA na platforme .NET.

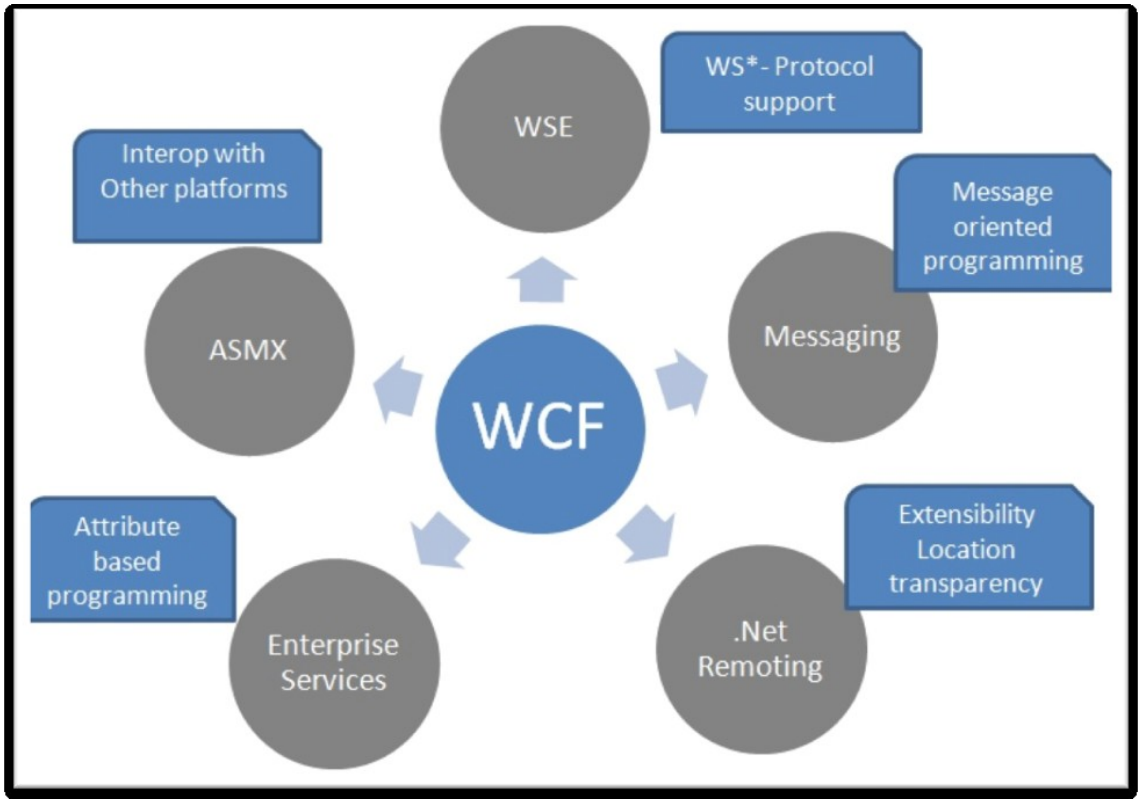
Obidva popísané typy poskytujú prostredie pre tvorbu webových služieb poskytujúcich nám základné črty SOA. Využívajú rôzne štandardy, ako SOAP, REST, HTTP a XML, čím nám zabezpečujú komunikáciu s heterogénnymi systémami, čo sa týka platformy. K publikovaniu svojich služieb používajú WSDL. Vzhľadom k tomu, že tieto možnosti spĺňajú požiadavky uvedené na začiatku 2. kapitoly, môžeme si vybrať, ktorú cestu si zvolíme. V tejto práci sme si zvolili MS .NET WCF. Alternatívy a spôsoby poskytnuté týmto frameworkom si popíšeme a rozanalyzujeme v podkapitole 3.1.

3.1 Technológie pre SOA v prostredí .NET

Pre tvorbu SOA architektúry sme si vybrali MS .NET WCF. Vlastnosťami, ktorými disponuje, vhodne zapadá do budovania nášho IDS. Pre lepšie pochopenie si objasníme pojem WCF, a taktiež si predstavíme možnosti, ktorými táto technológia disponuje.

3.1.1 Čo je WCF

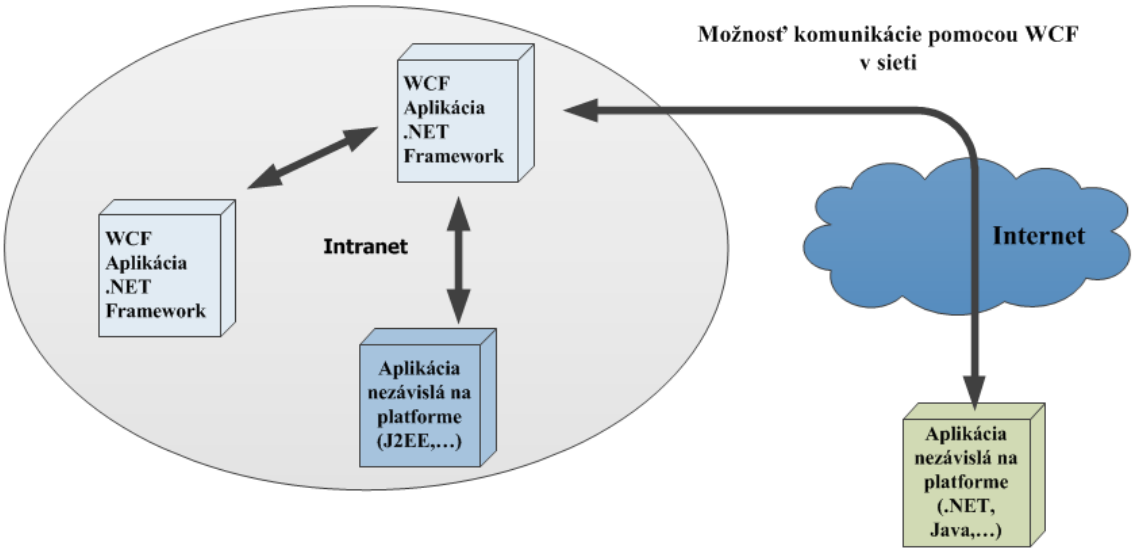
Akronym WCF znamená Windows Communication Foundation. Ide o jeden zo spôsobov, ako efektívne vytvoriť poskytovateľa disponujúceho webovými službami, kde dané služby využíva rôzna komunita klientov. Umožňuje definovať komunikačný charakter medzi klientom a serverom, pre ich vzájomnú interaktivitu. Technológia MS .NET WCF posunula vývoj servisne orientovanej komunikácie do novej dimenzie. Tento framework vyvinula spoločnosť Microsoft za účelom budovania aplikácií, ktoré sú schopné navzájom komunikovať v distribuovanom prostredí. WCF zastrešuje viacero technológií, ako ASMX webové služby, .NET remoting, WSE, Enterprise Service, resp. MSMQ.



Obrázok 6: Technológie zahrnuté v WCF frameworku[5]

WCF bol prvýkrát začlenený do .NET Framework 3.0. Zabezpečuje interoperabilitu s aplikáciami, fungujúcimi na rôznych platformách, resp. s aplikáciami, ktoré nie sú založené na WCF technológii. Taktiež poskytuje rôzne protokoly, ktoré zaisťujú bezpečnosť, spoľahlivosť, transakciu prenášaných dát, široký výber kódovania a transportu. Cieľom spoločnosti Microsoft bolo vytvorenie programovej platformy za účelom budovania, konfigurovania a nasadzovania sieťovo distribuovaných služieb. Aplikácie, ktoré sú založené na spomínanej technológii, sú ľahko udržovateľné, znova spustiteľné a rozšíriteľné. Poskytuje nám efektívnu alternatívu pre vytváranie desktopových, resp. webových aplikácií. Základným charakteristickým prvkom WCF je vytváranie slabo prepojiteľných elementov. Snaha zachovať klasické črty SOA architektúry, ktorými sú schopnosť opätovného použitia softvérových výhod a poskytovanie funkcionality týchto výhod v podobe služieb.

[5] Zdroj: http://r4r.co.in/WCF/01/tutorial/basic/Introduction%20of%20WCF.shtml



Obrázok 7: Možnosť komunikácie pomocou WCF v sieti

Budovanie servisne orientovanej architektúry v oblasti WCF poskytuje viacero alternatív. V minulosti medzi predchodcov tvorby webových služieb v.NETe patrili klasické webové služby (ASMX services). Súčasnosť poskytuje technológiu WCF, ktorá je zameraná na tvorbu servisne orientovanej komunikácie a zahŕňa aj spomínané ASMX služby (pozri obrázok 6). Pri práci s touto novšou technológiou sa zamýšľame nad správnou voľbou stratégie pre dolovanie dôležitých informácií zo systému prostredníctvom služieb. Aký je najefektívnejší spôsob získavania potrebných prostriedkov, ktoré sú následne poskytnuté klientovi? Medzi možné alternatívy patria spomínané webové služby (ASMX services), klasické WCF (WCF core), WCF Data Services, resp. WCF RIA Services. Presný spôsob výberu najvhodnejšej možnosti pre jednotlivú .NET aplikáciu neexistuje. Podrobná analýza charakteristických vlasností spomínaných typov nás môže nasmerovať k správnemu výberu. V nasledujúcom texte sa budeme venovať týmto stratégiám.

Komplexnejší pohľad na WCF poskytuje [4], resp. zdroj [5]. Ak si chceme ujasniť tvorby webových služieb pomocou dostupných technológií na platforme .NET, môžeme čerpať pomocou [6].

3.1.2 Web Services (ASMX)

Klasické webové služby (ASMX služby) vznikli ešte pred technológiou WCF, avšak postupom času sa stali jej súčasťou (pozri obrázok 6). Ide o dátový prístup založený na rozhraní. Predstavujú programovateľnú aplikačnú logiku, ktorá je dostupná prostredníctvom štandardných webových protokolov. K tým najvýznamnejším a najdôležitejším patrí SOAP protokol. SOAP využíva XML pre formátovanie a HTTP protokol pre transport požadovaných dát. Klienti, ktorí pristupujú k webovej službe, nepotrebujú vedieť nič o jazyku, resp. platforme, kde bola daná služba implementovaná, zaujíma ich akým spôsobom môžu so službou komunikovať a vymieňať si s ňou správy. V tomto prípade, ako sme už spomenuli sú tieto správy vo forme

SOAP protokolu. Toto nám zabezpečuje interoperabilitu, ktorá je mimoriadne žiadaná v mnohých prípadoch. Zjednodušený princíp komunikácie medzi poskytovateľom služby a klientom, ktorý k nej pristupuje , je zobrazený na obrázku 5.

Tento typ stratégie je vhodný, ak webová služba má spolupracovať s klientom, ktorého platforma nie je podstatná. Môže ísť buď o WCF klienta alebo nie. Už sme sa zmienili, že ASMX služby zaručujú interoperabilitu. V každom prípade je to jediné riešenie, ak váš server nepodporuje .NET 3.0, resp. neskoršiu verziu .NET Framework.

Používanie webových služieb musí byť dôsledné. Dôvodom môže byť, že služba nebude podporovať interoperabilitu. Príkladom môže byť, že ASMX služba vracia objekt typu *DataTable*, ktorý je podporovaný iba na platforme .NET. V prípade Silverlightu daný dátový typ nie je podporovaný, takže nevie, ako má s ním manipulovať. Ak od vašej aplikácie očakávate podporu transakcií, manažment instanciovania, súbežnosť biznis procesov a iné prvky, môžete byť dosť sklamaní. Tieto prvky nie sú zakomponované do klasických webových služieb. Spôsob, akým je možné budovať ASMX webové služby, je možné nájsť v prílohe v časti A.1.

Postupom času vznikajú nové protokoly a štandardy, ktoré touto funkcionalitou disponujú a sú v tomto smere určite lepším riešením. Medzi technológie obohatené o tieto črty patrí napríklad aj WCF.

3.1.3 WCF (core)

Ako bolo spomenuté, WCF technológia sa stala prvýkrát realitou v .NET Framework 3.0. Umožňuje nám vytvárať, nielen samotné webové služby, ale taktiež klientov, ktorí pristupujú k službám. Služby, ktoré vytvárame, sme schopní veľmi ľahko adaptovať rôznym klientom. Kódovanie medzi klientom a serverom je v podobe XML, binárnej podobe alebo MTOM (využívaný pri veľkom množstve prenášaných dát). Samotná komunikácia prebieha medzi koncovými bodmi (*endpointami*). Každý *endpoint* sa viaže s určitom adresou, ktorá definuje, kde sa služba nachádza. Jedna služba môže disponovať viacerými *endpointami*. Väzba (*binding*) nám špecifikuje ako s danou službou komunikovať. Pri deklarovaní väzby sa musíme zamýšlať, s akým klientom chceme komunikovať. Ak klient nie je založený na WCF, je potrebné zabezpečiť interoperabilitu.

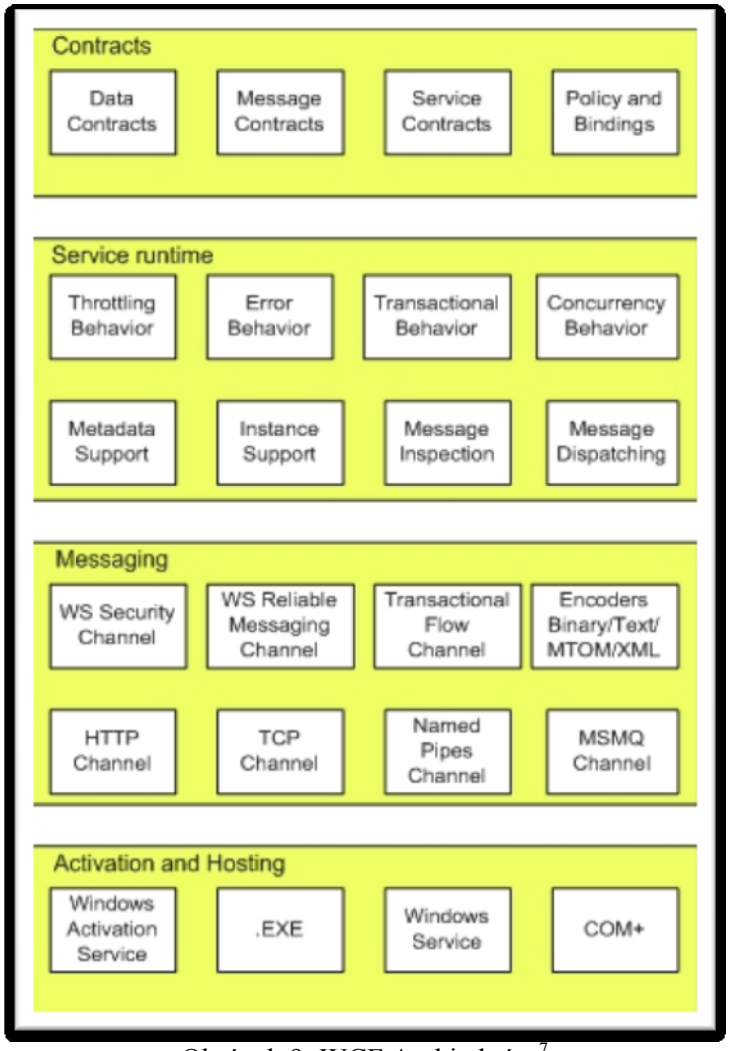
Obrázok 8: WCF ABC (A-Adresa, B-Väzba, C-kontrakt)[6]

Na rozdiel od webových služieb ASMX, WCF služby môžu komunikovať prostredníctvom rôznych protokolov, nielen pomocou HTTP. Na výber sa nám ponúkajú ďalšie, ako napríklad TCP, Named Pipes, resp. MSMQ. HTTP nám ponúka možnosť dvoch druhov väzby medzi koncovými bodmi (*endpointami*). Prvý typ väzby je *BasicHttpBinding*, ktorý je typický pre webové služby ASMX. Zabezpečuje nám funkčnosť nezávislú na platforme. Je realizovaný pomocou HTTP a SOAP. Druhý typ je *WsHttpBinding*, ktorý taktiež umožňuje interoperabilitu, ale navyše podporuje transakcie, bezpečnosť a *sessions*.

V prípade vytvárania jednolivých služieb, ktoré predstavujú základný konštrukčný prvok servisne orientovanej architektúry, je nutné definovať kontrakt, ktorý bude služba implementovať. Ako bolo spomenuté, kontrakt služby popisuje, akými operáciami služba disponuje. Na základe tohto faktu môžeme konštatovať, že základná podoba WCF je metóda dátového prístupu založená na rozhraní. Aby požadované rozhranie fungovalo ako kontrakt služby, je potrebné aplikovať atribút *ServiceContract* k danému rozhraniu, ktoré predstavuje službu. Týmto spôsobom vo WCF oddeľujeme definíciu služby od jej implementácie. Operácie, ktorými má služba disponovať, obohatíme o atribút *OperationContract*. Naopak tým, ktoré nemajú presahovať za hranice systému, nepriradíme spomínaný atribút. Konkrétny príklad použitia sa nachádza v prílohe v časti „Ako tvoriť webové služby pomocou klasického WCF“.

Služby neprezrádzajú implementáciu technológií za svoje hranice. Podobne WCF nedovoľuje vystavovať CLR dátové typy (napr.: *Boolean*, *Int32*, *String*, a pod.) za hradby služby. To, čo je nutné zrealizovať, je konvertovať dátové typy z a do štandardnej neutrálnej podoby. Táto podoba je jednoduchá XML schéma. Spôsob, akým sa táto konverzia rieši, je formálna špecifikácia nazývaná *DataContract*. *DataContracty* sú publikované v metadátach služby, následne vďaka ním klient skonvertuje túto neutrálnu podobu do podoby, s ktorou je schopný pracovať. Dátové typy týchto objektov musia byť označené atribútom *DataContract* a jednotlivé členy dátového typu atribútom *DataMember*, inak budú ignorované pri serializácii. Pre názornú ukážku slúži opäť text v prílohe s názvom „Ako tvoriť webové služby pomocou klasického WCF“.

[6] Zdroj: http://blogs.msdn.com/b/paolos/archive/2009/11/17/customizing-and-extending-the-biztalk-wcf-adapters.aspx



Obrázok 9: WCF Architektúra[7]

Táto metóda sa uprednostňuje v prípade, ak si projekt vyžaduje veľmi podrobnú kontrolu biznis procesu v aplikácii. WCF je založené na mnohých štandardoch, ktoré nám umožnia napasovať správne ingrediencie s rôznymi klientami. Existuje tu podpora pre SOAP aj pre REST. Navyše je rýchlejšia oproti klasickým ASMX webovým službám.

Nevýhodou je minimálna verzia .NET Framework 3.0. na serveri. Komplikovanejšie spracovanie WCF REST služieb.

3.1.4 WCF Data Services

WCF Data Services predstavujú veľmi tenkú vrstvu medzi URI modelom a prístupom k dátam za pomoci HTTP protokola. Umožňujú budovanie webových služieb, ktoré na rozdiel od

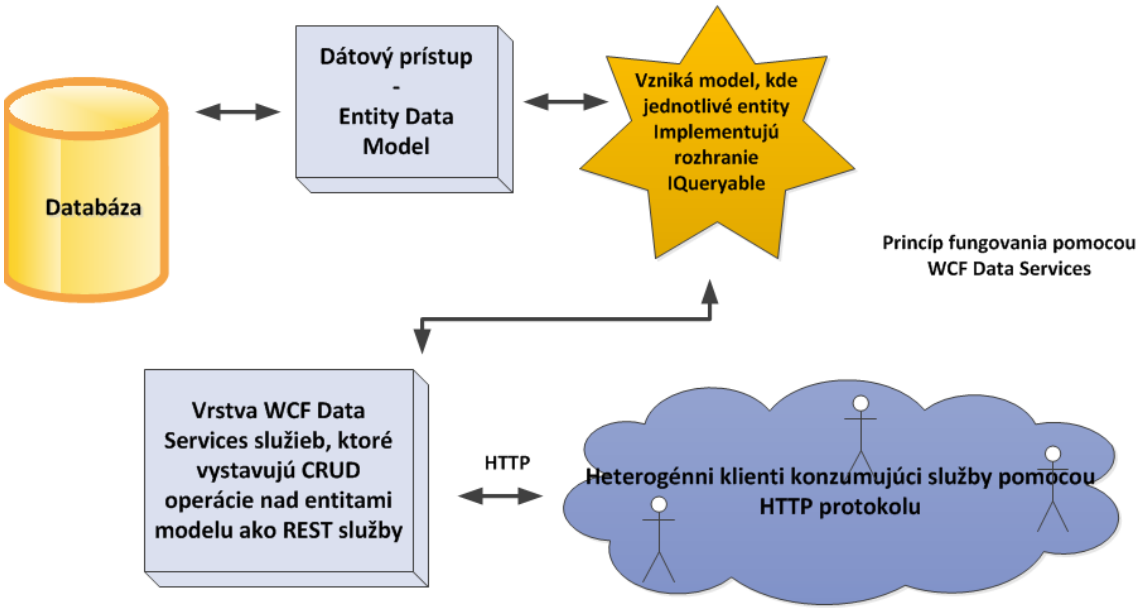
[7] Zdroj: http://www.infoq.com/articles/net-service-registry

predošlých typov, ktoré využívali prevažne SOAP protokol, sú postavené na OData protokole. OData protokol sa snaží všetky zdroje, ktoré majú byť publikované službami, jednoznačne identifikovať pomocou URL. Pri posielaní správ medzi klientom a serverom je možné používať formát typu Atom, ktorý je založený na XML alebo JSON. Formát Atom nie je veľmi obohatený o dotatočné informácie, takže sa prenáša menší objem dát ako pri SOAP protokole, čo znamená zlepšenie výkonu.

Príklad prístupu k dátam (zdroju) pomocou URL:

```
http://localhost:4628/ProductService.svc/product/25
```

WCF Data Services predstavuje komponentu .NET Frameworka, ktorá sa opiera o Entity Data Model, kde využíva vzťahy a prepojenia medzi jednotlivými entitami. Preto je veľmi vhodné túto komponentu využívať práve v spojitosti s týmto modelom. Avšak je možné produkovať zdroje aj z CLR tried, kde objekty vracajú inštanciu, ktorá implementuje rozhranie *IQueryable*. Inými slovami, je možné si vytvoriť vlastný zdroj dát, nezávislý od Entity Framework.



Obrázok 10: Princíp fungovania pomocou WCF Data Services

WCF Data Services sú správnou voľbou, ak pracujeme s ADO.NET Entity Framework dátovým modelom. Pracujeme nad jednoduchým a známym HTTP protokolom. Znalosť ostatných štandardov nie je potrebná. Neformulujeme rozhranie na strane servera. Rozhraním je URL adresa, ktorou pristupujeme k zdrojom. Z tohto dôvodu je veľkou výhodou, že nie je potrebné modifikovať rozhranie na strane servera a následne aktualizovať webovú službu. Pomocou zostavenia *System.Data.Services.Client* sme schopní formulovať dotazy na strane klienta. Táto skutočnosť nám dáva väčšiu voľnosť pri pristupovaní k zdrojom. Taktiež sa

vyhneme namáhavej práci rozboru kódu odpovede, ktorú nám konvertuje do čitateľnej podoby. Klientov je preto vhodné formulovať do podoby dopytovateľných koncových bodov. Príkazy sú ľahko formulované pomocou LINQ technológie. Príklad je uvedený na výpise 1:

```
var result = from roadElement in _carDBEntities.RoadElements
             where roadElement.roadId.Equals(1000)
             select roadElement;
```

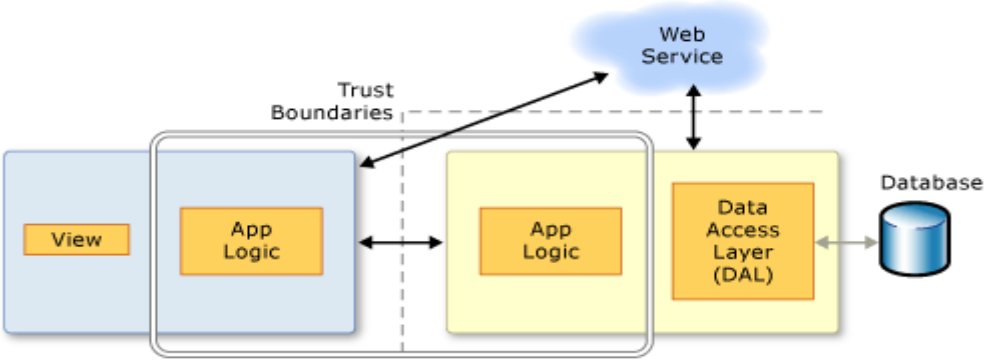
Výpis 1: LINQ príkaz pre získanie dát zo serveru

Podrobnejšie informácie ohľadom spolupráce LINQ technológie s webovými službami je možné čerpať z [7].

Ak požadujeme pevnú kontrolu nad rozhraním služby, tak táto metóda nie je pre nás ideálnym riešením. Respektíve, ak chceme mať väčší prehľad o dátach, ktoré putujú medzi klientom a databázou. Rozšírenie vedomostí o WCF Data Services je možné nadobudnúť pomocou [8].

3.1.5 WCF RIA Services

Tento framework bol vytvorený so zámerom zjednodušiť vývoj RIA aplikácií. Vývoj a koordinácia aplikačnej logiky na serveri a na klientovi bola sčasti obtiažna, bolo potrebné samostatne aktualizovať klientskú stranu. Pomocou WCF RIA Services dochádza k veľmi úzkej prepojiteľnosti logiky medzi prezentačnou a strednou vrstvou. Kód klienta je vždy automaticky aktualizovaný po skompilovaní spomínanej strednej vrstvy na serveri. Táto vrstva spravidla pozostáva z webového aplikačného projektu, ktorý obsahuje webové služby. Vďaka prepojeniu medzi vrstvami dochádza ku generovaniu kódu na klientovi. Kód, ktorý je automaticky generovaný, môže predstavovať napríklad triedy z dátového zdroja, autentifikáciu, resp. validačnú logiku.



Obrázok 11: Miesto zamerania sa WCF RIA Services pri vývoji viacvrstvovej aplikácie[8]

[8] Zdroj [9]

Pri sumarizovaní tohto spôsobu vývoja SOA pomocou WCF technológie, je nevyhnutné vyzdvihnúť zjednodušenie vývoja aplikačnej logiky medzi strednou a prezentačnou vrstvou.

Na ďalší pozitívny aspekt WCF RIA Services sa pozrieme v ďalších kapitolách. RIA Services boli primárne vytvorené pre prácu so Silverlightom. Jedná sa o metódu, ktorá je založená na definovaní rozhrania na strane servera. Základné informácie o WCF RIA Services poskytuje [9].

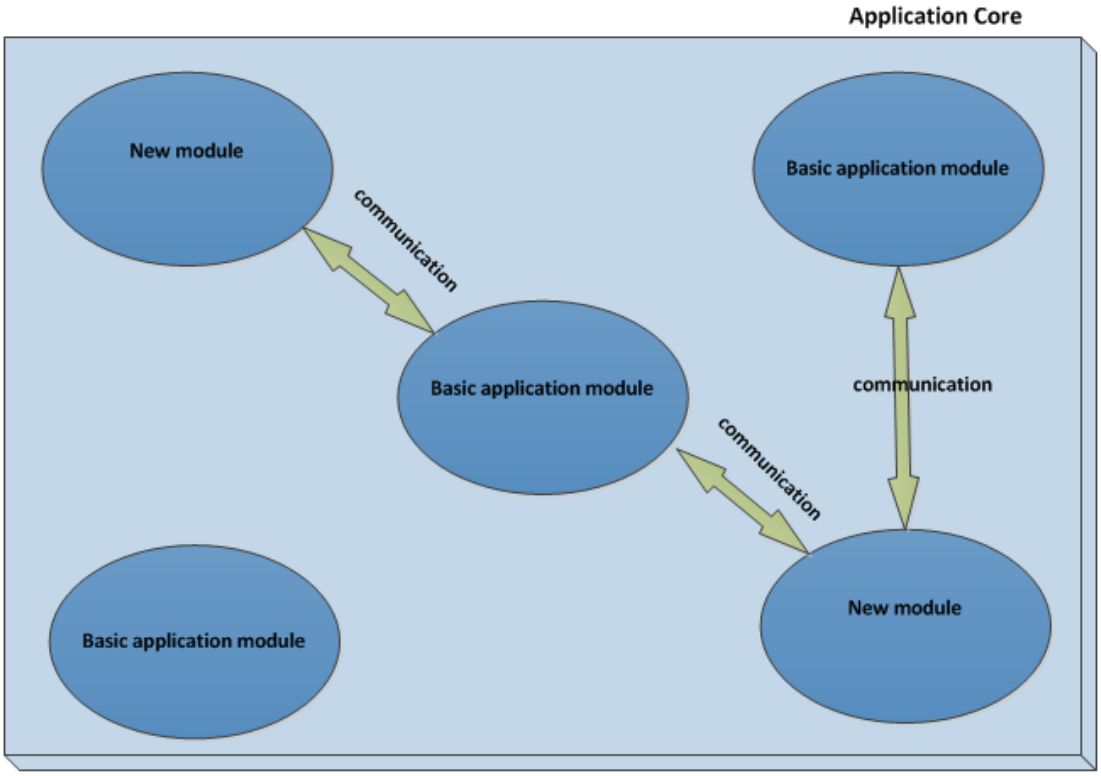
3.2 Výber aplikačného modelu pre budovanie SOA

Použitá architektúra bude zahŕňať Silverlight aplikáciu. Silverlight je framework, ktorý bol špeciálne vytvorený na tvorbu RIA aplikácií. Keďže sa chceme uberať práve týmto smerom, SOA architektúra sa bude opierať o WCF RIA Services služby. Výhody, ktorými disponuje vybraný framework, vhodne zapadajú do kombinácie so Silverlight klientom. Ich popis je v kapitole 3.1.5. Produkovaná aplikácia nie je až tak rozsiahla, čo môže byť ďalším pozitívnym faktorom pre spomínaný výber. Na druhej strane nejde o framework, ktorý by bol ostrieľaný z každého uhla pohľadu. Predsa len ide o novší produkt firmy Microsoft, a preto sa môžeme stretnúť so situáciami, ktoré nebudú adekvátne vyhovovať našim predstavám.

4 Popis architektúry navrhovaného systému

Štvrtá kapitola práce popisuje architektúru systému do podrobnejších detailov. Prezentuje jej základné črty a vlastnosti, ktorými sú otvorenosť, živosť, a schopnosť komunikovať v spolupráci s dodatočnými modulmi, resp. systémami. Pri vývoji bola snaha rozdeliť systém ako celok na jednotlivé moduly, ktoré vhodne do seba zapadajú a spolupracujú. Štruktúra aplikácie by mala umožniť jednoduché zapracovanie novej komponenty za účelom zvýšenia funkcionality výsledného produktu.

Na jadro systému pozeráme ako na čiernu skrinku, ktorá by mala logicky komunikovať s rôznymi typmi klientov, a taktiež by mala byť schopná pracovať s rozličnými dátovými servermi poskytujúcimi dáta. Ilustrácia na obrázku 12 vykresľuje architektúru jadra systému zloženú z modulov, ktorú je možné doplniť o dodatočné komponenty. Architektúra poskytuje prostredie, kde sa nová komponenta ľahko adaptuje a je schopná pracovať.



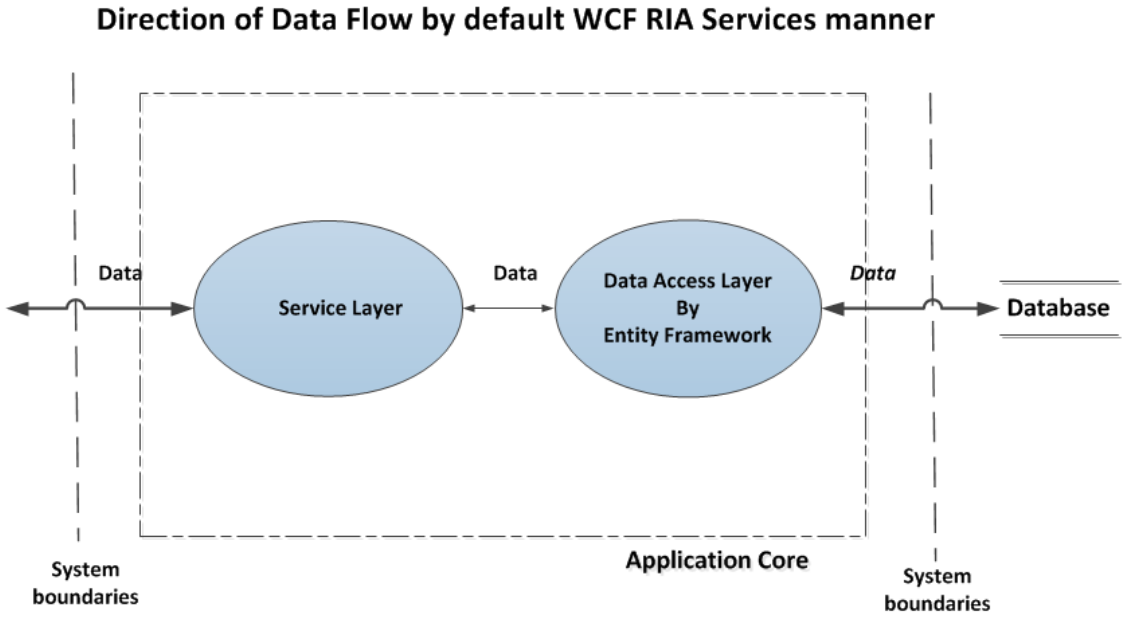
Obrázok 12: Jadro informačného systému zostavené zo základných vrstiev, doplnených o nové moduly

4.1 Štruktúra jadra aplikácie

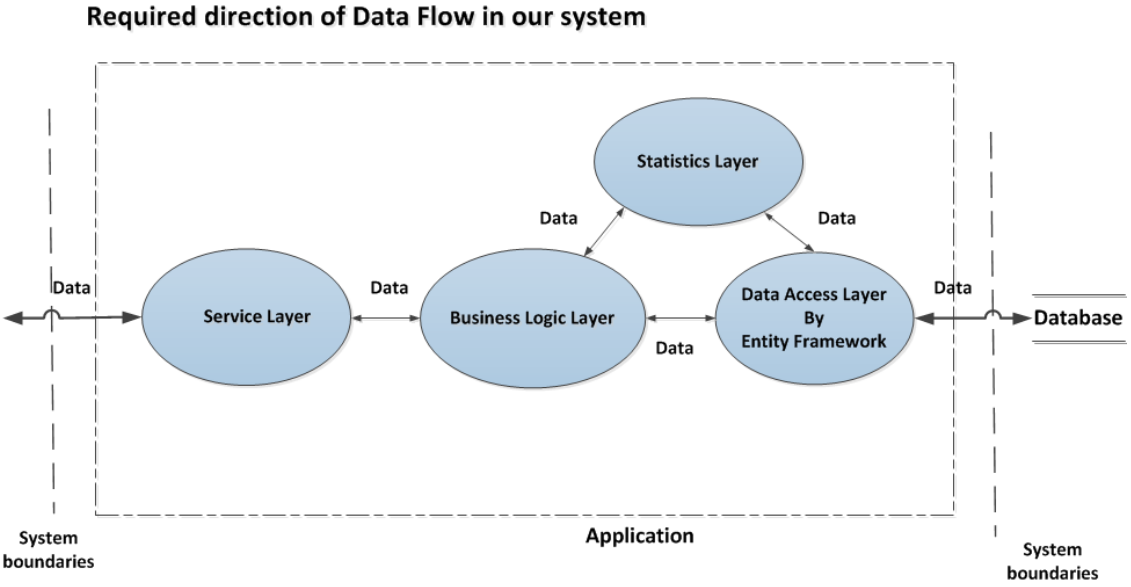
V kapitole 3.2 sme vybrali WCF RIA Services, ako preferovaný spôsob tvorby SOA. Servisná vrstva by mala predstavovať vstupný bod pre klientov, ktorí chcú využívať možnosti systému, a takisto aj pre dáta putujúce do a zo systému. Za použitia WCF RIA Services je práve tento modul ten, ktorý je pomocou *RIA Link* väzby prepojený so Silverlight klientom.

V prílohe s názvom „Ako tvoriť webové služby pomocou WCF RIA Services" ukazujeme ako vytvoriť jednoduchú službu, ktorá sa opiera o EF technológiu, kde pri nastavovaní dátového zdroja služby priamo vyberáme určitú triedu (*CarDatabaseEntities*). Avšak, my pri vytváraní jednotlivých služieb nebudeme nastavovať žiadny zdroj a ponecháme ho prázdny. Týmto spôsobom bude každá služba dediť z triedy *DomainService*. Možno sa tu vynára otázka „Prečo si komplikovať prácu, keď to môže byť automaticky generované?". Dôvodom pre toto rozhodnutie je získanie väčšej kontroly nad samotným tokom dát. Takto poskytneme systému možnosť výberu požadovaného toku dát v biznis procese. Namiesto priameho prístupu k dátovému zdroju, sa požaduje lepšia manipulácia toku dát, ktoré idú do a z fyzického dátového zdroja. Možnosť zapojenia logickej vrstvy, resp. nového modulu do biznis procesu je jednou z hlavných príčin tohto rozhodnutia. V úzkej komunikácii so servisnou vrstvou je logická vrstva, ktorá manipuluje s entitami a posúva ich do požadovaných smerov. Pri tomto

probléme by do vývoja systému vhodnejšie zapadalo klasické WCF, ktoré má väčšiu kontrolu nad tokom dát. Avšak z dôvodu spolupráce so Silverlight klientom, sme zvolili WCF RIA Services. Rozdiel toku dát vo vyvíjanom systéme a pôvodným zámerom pomocou technológie RIA Services je zobrazený na obrázkoch 13 a 14.



Obrázok 13: Pôvodný (zjednodušený) zámer toku dát pomocou WCF RIA Services

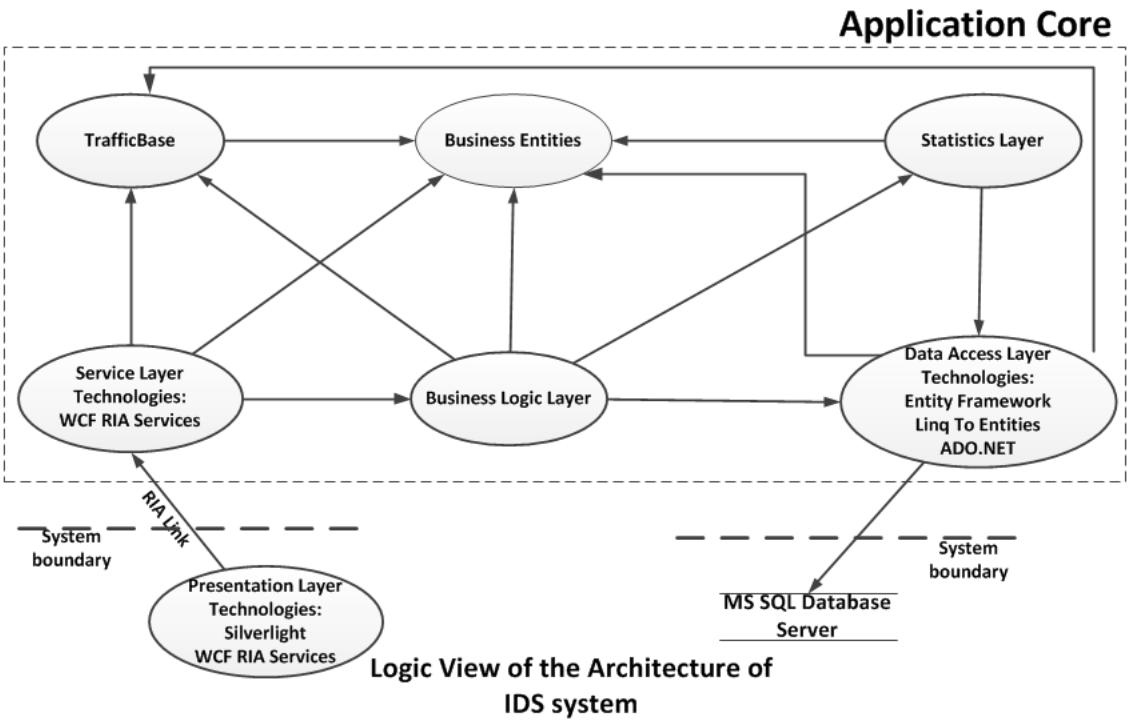


Obrázok 14: (Alternatívny) smer toku dát vo vyvíjanom systéme

4.2 Rozšírenie architektúry o nový modul

Jedna z komponent, ktorú je potrebné zakomponovať do aplikácie je štatistická vrstva. Tento dodatočný modul vykonáva štatistickú analýzu. Z hľadiska našich požiadaviek na systém, disponuje vysoko požadovanou funkcionalitou, a preto je ho nevyhnutné začleniť do štruktúry. Bez tohto modulu by neboli poskytnuté štatistické výsledky, o ktoré sa aplikácia nosne opiera a závisí na nich.Na paralelnom vývoji tejto komponenty sa podieľal tímový kolega. Komponenta bola iterovane začleňovaná do budovaného produktu. V štruktúre systému je prepojená s logickou vrstvou a vrstvou dátového prístupu. Poskytuje sofistikované výpočty neurónových sietí a pod. Bližšie informácie o nej sú dostupné na [15].

Architektúra systému poskytuje vhodné prostredie pre jednoduché zapracovanie nového modulu do celkovej štruktúry produktu. Pre lepšiu predstavu je architektúra na nasledujúcom obrázku 15, nielen doplnená o nový modul, ale aj o konkrétny typ klienta a DB server. Výber a popis typu klienta je bližšie popísaný v kapitole 5.2. Náčrt je obohatený o závislosti medzi jednotlivými modulmi. Ako dodatočná informácia sa u niektorých uvádza technológia, ktorá bola pri ich implementácii použitá.



Obrázok 15: Pohľad na architektúru vytváraného informačného systému s novým modulom štatistiky

Z predchádzajúcej ilustrácie môžeme vidieť, že nový modul je schopný spolupracovať so základnými modulmi jadra systému. Konkrétne ide o logickú vrstvu a o vrstvu dátového

prístupu. Pri ich komunikácii je potrebné si ujednotiť spôsob, akým budú spolupracovať. Táto komponenta je príkladom, kde komunikácia medzi zapracovaným modulom a jadrom systému prebieha v obidvoch smeroch . Nasledujúci výpis 2 prezentuje jednoduché využitie funkcionality nového modulu v zdrojovom kóde:

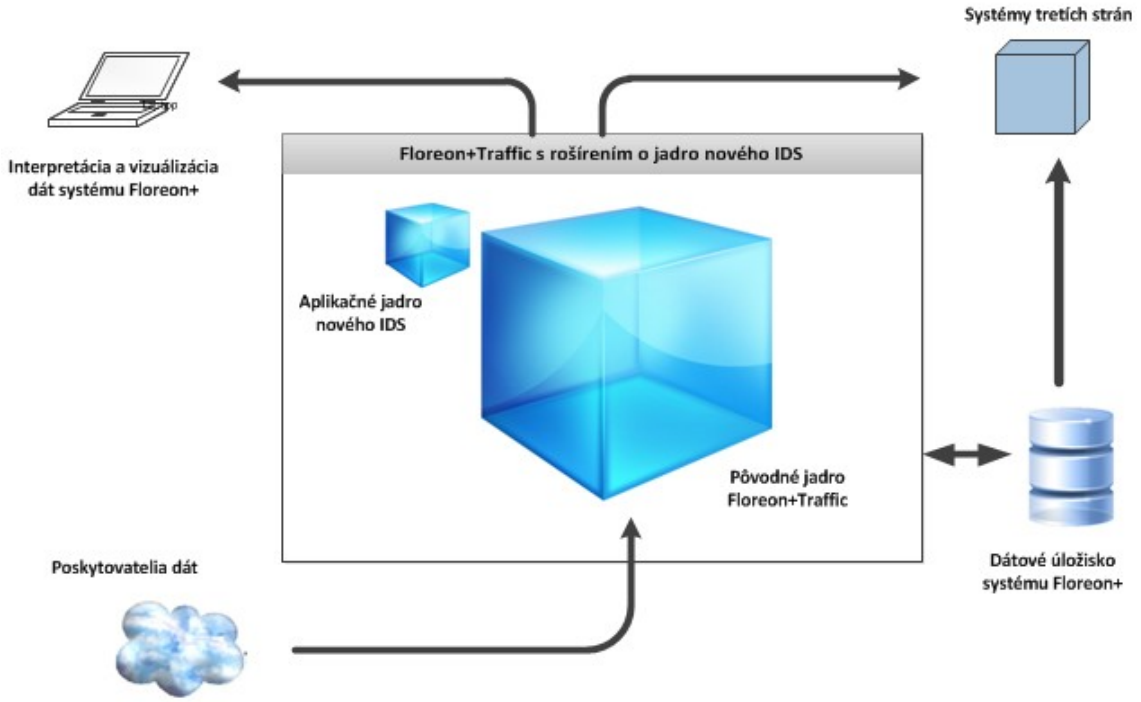
```
public Dictionary<DateTime, double> GetHistoricalLinearRegression(DateTime pFromDate,
    DateTime pToDate, double pLinkId)
{

  Dictionary<DateTime, double> result =
      Statistics.GetPrediction(PredictionType.HistoricalLinearRegression,
      (long)pLinkId, pFromDate, pToDate);

    return result;
}
```

Výpis 2: Využitie nového štatistického modulu pri implementácii

4.3 IDS ako modul väčšieho systému

Architektúra nového IDS by mala nadobudnúť podobu s perspektívou rozšírenia iných systémov. Jadro IDS by malo byť ľahké začleniť do väčšej architektúry, čím by náš produkt získal na kvalite, ohodnotení, a tiež možnosti využitia. Samotné jadro aplikácie by malo predstavovať modul, ktorého zapracovaním do robustnejšieho systému rozšírime funkcionalitu daného produktu. Avšak, je nutné sa zmieniť, že výsledný produkt je schopný fungovať samostatne.

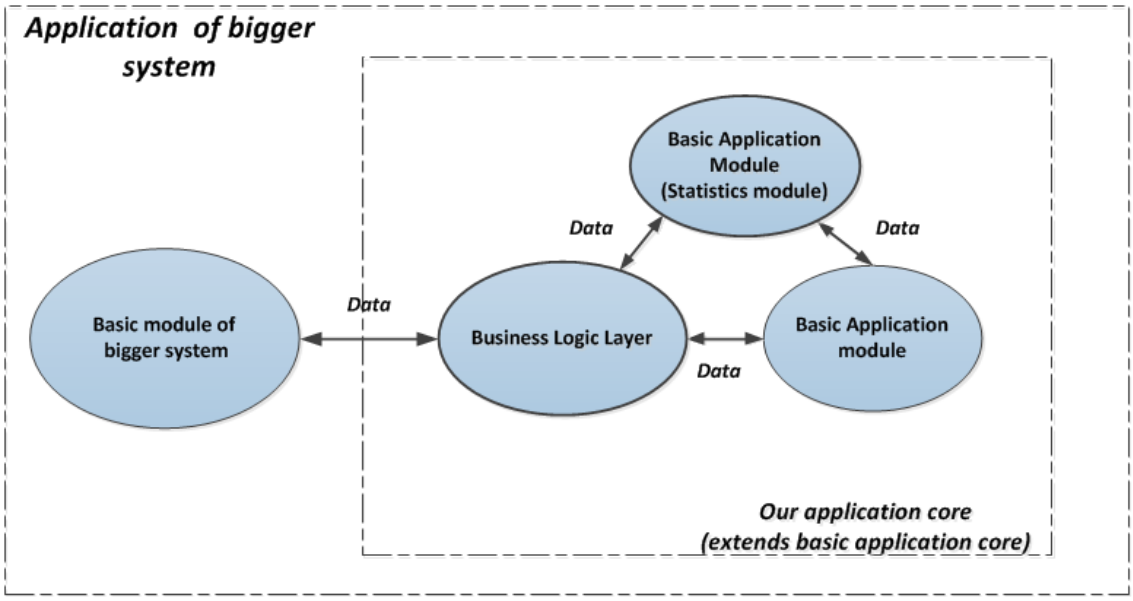
Floreon+Traffic poskytuje vhodné prostredie, ku ktorému by sme mohli adaptovať nami vytváraný systém. Základná myšlienka pri budovaní nášho IDS sa taktiež opiera o dáta, nad ktorými sa vykonávajú algoritmy poskytujúce užitočné výsledky. Dátový zdroj je rovnaký pre obidva produkty, teda majú rovnakú dátovú štruktúru, s ktorou môžeme efektívne pracovať. Týmto spôsobom sme schopní vytvoriť nadstavbový modul k systému Floreon+.



Obrázok 16: Zabudovanie jadra nového IDS do systému Floreon+Traffic

Produkt Floreon+Traffic dokáže efektívne zbierať dáta. Za pomoci nami vytváranej aplikácie, je možnosť rozšírenia funkcionality o štatistické výsledky, resp. o iné funkcie, ktoré by boli doplnené do našej aplikácie. Systém, ktorý by využíval túto aplikáciu by mohol využívať funkcionalitu pre interné účely, alebo poskytovať výsledky nového modulu pre potreby klientov.

Princíp komunikácie našej aplikácie, ktorá v tomto prípade predstavuje modul, so systémom do ktorého je zabudovaná, je zobrazený na nasledujúcom obrázku 17. Dôležitým miestom je spolupráca medzi logickou vrstvou a modulom väčšieho systému, ktoré sú spolu prepojené.



Obrázok 17: Princíp zabudovania a komunikácie medzi našou aplikáciou začlenenou do väčšieho systému

5 Realizácia požadovanej funkcionality

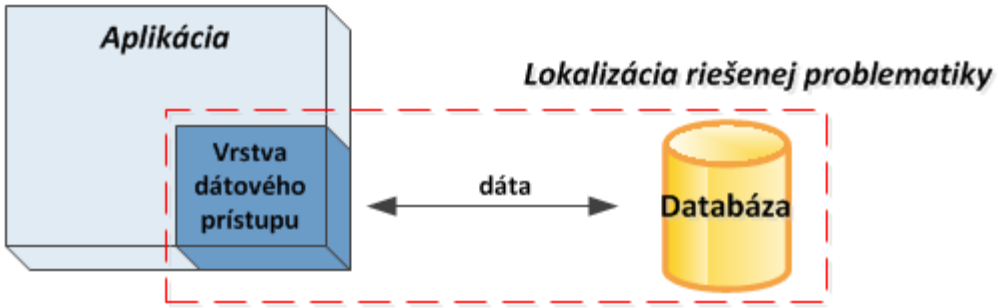
V nasledujúcej kapitole si rozoberieme dôležité prvky, technológie a spôsoby, ktoré zabezpečujú požadovanú funkcionalitu systému. Ten by mal byť konkurencieschopný a atraktívny z pohľadu užívateľa, ale taktiež by mal disponovať základnými funkciami ako jeden zo znakov kvality produktu.

V nasledujúcich podkapitolách sa pozrieme na niektoré z dôležitých modulov systému, o ktoré sa výstavba softvérového diela opiera. Ich postavenie a funkciu v systéme si priblížime prostredníctvom častí programovacieho kódu v jazyku C#, kde sa miestami stretneme s možnými problémami a ich riešením.

5.1 Prístup k dátovému zdroju a spolupráca s DB serverom

Pri pohľade na architektúru vidíme, že aplikácia by mala mať prístup k databázovému serveru využívajúc reálne dáta. Požadovaný cieľ sa mal dosiahnuť využitím moderných a efektívnych technológií. Pre prístup k dátam je potrebné vytvoriť modul, ktorý bude plniť túto funkciu. Náš zdroj dát predstavuje vzdialená databáza, s ktorou tento modul musí komunikovať. Vo vytváranej aplikácii je nutné zabezpečiť prístup k dvom rôznym databázovým serverom. Typ databázového servera je v obidvoch prípadoch MS SQL Server 2008. Pre zvýšenie úrovne bezpečnosti systému je vhodné, aby implementovaný kód neobsahoval prístupové dáta na DB server, ale aby boli tieto informácie presunuté do konfiguračného súboru.

Pôvodne bola predstava, že pri vývoji systému a dolovaní dát z DB bude využitý iba framework ADO.NET Entity Framework za pomoci technológie LINQ To Entities. Avšak, postupná implementácia odhalila nedostatky spomínaného frameworku v spolupráci s databázovým serverom a bolo potrebné využiť aj klasický ADO.NET. Popis práce s nimi si priblížime v nasledujúcich riadkoch.

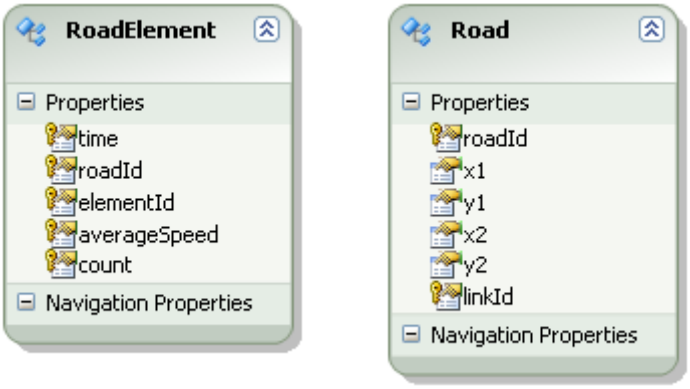


Obrázok 18: Lokalizovanie problematiky danej kapitoly 5.1

5.1.1 Primárny spôsob komunikácie s databázou

V minulosti programátor musel vynaložiť veľké úsilie, aby prepojil objekty v databáze s objektami v aplikácii. ADO .NET Entity Framework prináša jednoduchší spôsob spolupráce medzi dátovým modelom a programovacím jazykom. V súčasnosti nám táto technológia ponúka alternatívu, ktorou efektívne namapujeme objekty z DB do objektovo orientovaného jazyka. Z týchto dôvodov sa my v tejto práci naladíme na vlnu dátového prístupu pomocou popisovaného frameworku.

Po úspešnom vytvorení dátového modelu, kde vzniknutý súbor má formát *edmx*, máme voľnú cestu k definovaniu príkazov nad daným modelom. Spôsob tvorby dátového modelu pomocou EF sprievodcu, je popísaný v prílohe v časti „Tvorba dátového modelu pomocou Entity Framework". Využitie EF modelu v spolupráci s WCF Data Services je načrtnuté na obrázku 10.



Obrázok 19: Model vygenerovaný pomocou EF

Pre názornú ukážku prístupu k dátam nám poslúži výpis 3 pod týmto odstavcom, ktorý získava vďaka objektu *_carDBEntities* a jeho vlastnosti *RoadElements* všetky inštancie typu *RoadElement*. Objekt *_carDBEntities* predstavuje typ *CarDatabaseEntities*, ktorý je odvodený od triedy *ObjectContext*. Pomocou LINQ To Entities sme schopní pridať obmedzenia do príkazu, a tým špecifikovať naše požiadavky nad danými zdrojom dát. Pomocou operátora *join* spojíme dátové zdroje, ktoré spolu neasociujú, ale zdieľajú spoločný atribút. V ukážkovom príklade je to atribút *roadId*. V podmienkach, ktoré píšeme za klauzulu *where* sa požaduje, aby atribút *linkId* objektu *roadElement* bol rovný hodnote vstupného parametra *pIdRoad*. Druhou podmienkou je, aby objekt *time*, ktorý predstavuje vlastnosť objektu *roadElement* aspoň raz figuroval v zozname dátumov, ktoré vstupujú ako parameter pod názvom objektu *pDateTimes*. Nakoniec je záznam zoradený vzostupne pomocou výrazu *orderby*. Podobným spôsobom pracujeme aj pri dopytovaní iných potrebných dát.

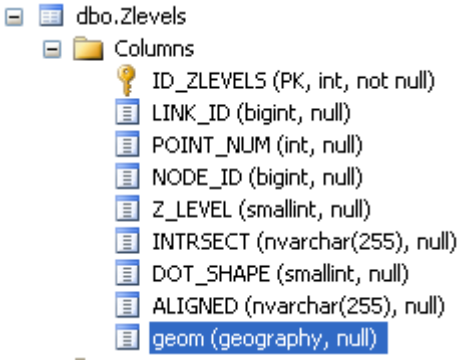
```
public List<RoadElementBE> GetRoadElementsByIdRoadByDateTimes(double pIdRoad,
    List<DateTime> pDateTimes)
{

var result = from roadElement in _carDBEntities.RoadElements
                join road in _carDBEntities.Roads on
                roadElement.roadId equals road.roadId
                where road.linkId == pIdRoad && pDateTimes.Contains(roadElement.time)
                orderby roadElement.time ascending
                select roadElement;
```

Výpis 3: Príkaz pre získanie dát za pomoci technológie LINQ To Entities

5.1.2 Práca s priestorovými dátami v databáze

Ak sa spätne vrátime k téme, ku ktorej sa snažíme vytvoriť informačný systém, uvedomíme si, že budeme musieť pracovať s priestorovými dátami. Priestorové dáta pracujú so

zemepisnou šírkou a zemepisnou dĺžkou, kde tieto súradnice jednoznačne identifikujú miesto na Zemi. Pri vývoji systému sa dostávame do kontaktu v databáze *Navteq* s tabuľkami, ktoré obsahujú atribúty typu *Geography*. MS SQL Server 2008 podporuje tieto dátové typy. Taktiež tieto typy disponujú niekoľkými veľmi užitočnými metódami, ktoré však EF nie je schopný rozpoznať a pracovať s nimi. Presne špecifikované, CLR nie je schopný rozlúsknuť tento dátový typ a pracovať s nim na platforme .NET. Ako riešenie sme zvolili klasický ADO.NET.



Obrázok 20: Tabuľka Zlevels obsahujúca atribút *geom* dátového typu *geography*

Ak je potrebné vysvetliť pojem ADO.NET, tak jeho význam si zľahka priblížime a popíšeme v tomto odstavci. ADO.NET je knižnica obsahujúca sadu tried, ktorá poskytuje prístup k fyzickému úložisku dát. Umožňuje komunikáciu s rozličnými dátovými servermi, za účelom získania dát. Disponuje rôznymi poskytovateľmi, ktorí slúžia na vytvorenie spojenia s databázou, resp. na vytvorenie dopytovacích príkazov, alebo uchovávanie výsledkov. Medzi nich patria .NET Framework Data Provider pre SQL Server, .NET Framework Data Provider pre OLE DB, alebo napríklad .NET Framework Data Provider pre Oracle. Táto knižnica disponuje triedami, ktoré sa buď zaoberajú komunikáciou medzi aplikáciou a DB, alebo ide o triedy, ktoré dočasne ukladajú výsledné dáta po vykonaní príkazov, ako napríklad trieda *DataSet*.

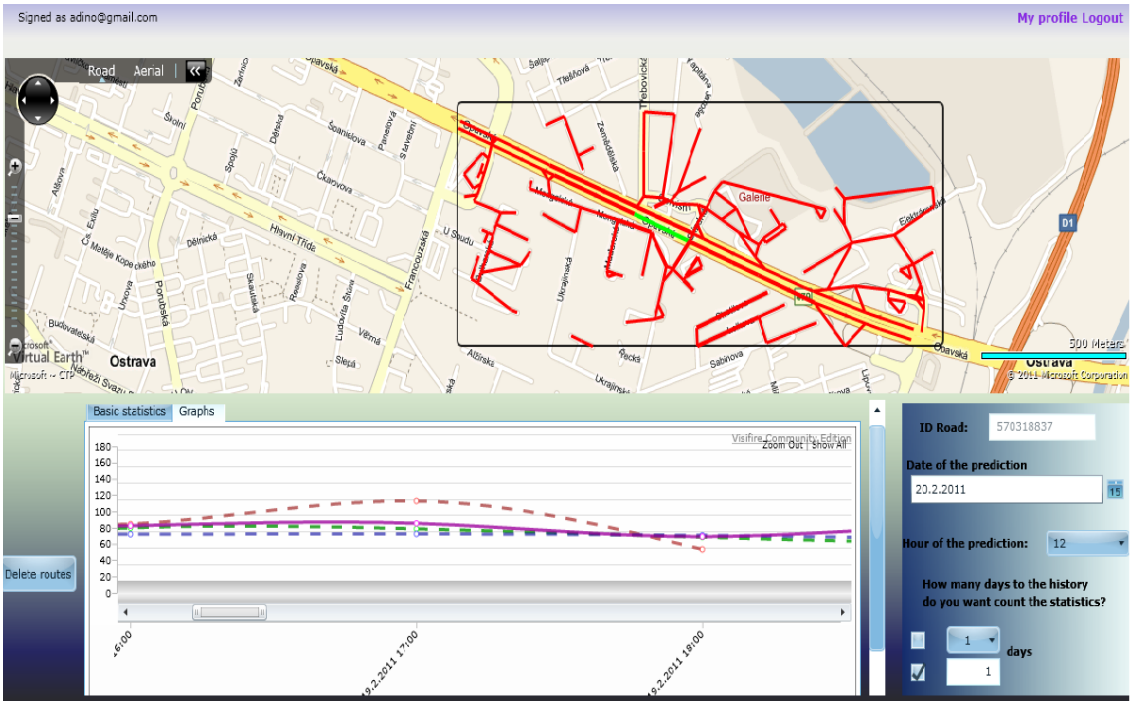
Na začiatku tejto podkapitoly sme si popísali dôvod, prečo sme sa obrátili na túto knižnicu. Alternatíva riešenia problému je pre nás pracnejšia, avšak výsledok bude spĺňať naše požiadavky. Za pomoci knižníc *System.Data.dll* a *Microsoft.SqlServer.Types.dll*, môžeme jednoducho v .NETe vytvoriť objekty typu *SqlGeography*. Tie následne pridáme do SQL príkazu a pomocou ďalších tried ADO.NET získame požadované dáta z fyzického dátového zdroja. Metódy, ktoré je možné vykonať nad priestorovými dátami, píšeme „natvrdo", čo však nie je najlepšia technika programovania. Formulovanie správne fungujúceho príkazu nad priestorovými dátami bolo inšpirované zdrojom [10]. Nasledujúci výpis 4 nám ukazuje spôsob vytvárania príkazu:

```
public NodeBE GetTheNearestRoadFromPoint(string pConnStr, double pLatitude, double pLongitude)
{
    double bufferSensibility = 20;

    SqlString sqlString = new SqlString(String.Format("POINT({0} {1})",
        pLongitude.ToString().Replace(',', '.'), pLatitude.ToString().Replace(',', '.')));
    SqlChars sqlChars = new SqlChars(sqlString);
    SqlGeography selectedPoint = SqlGeography.STPointFromText(sqlChars, 4326);

    SqlGeography areaOfPoints = selectedPoint.STBuffer(bufferSensibility);

    string query = String.Format(@"SELECT TOP 1 s.NODE_ID, s.LINK_ID, s.geom.Lat AS Lat,
    s.geom.Long AS Long, s.geom.STDistance('{1}') AS Distance FROM Zlevels s
    WHERE s.geom.Filter('{0}') = 1 AND s.geom.STIntersection('{0}').STNumGeometries() <> 0
    AND s.NODE_ID <> 0 ORDER BY s.geom.STDistance('{1}') ASC", areaOfPoints, selectedPoint);

    using (SqlCommand sqlCommand = new SqlCommand(query, new SqlConnection(pConnStr)))
```

Výpis 4: Tvorba príkazu pomocou ADO.NET a *SqlGeography* nad priestorovými dátami

5.2 Budovanie klienta IDS

V dnešnej dobe je zrejmé, že webová aplikácia musí užívateľa svojim rozhraním zaujať. RIA aplikácie dnes poskytujú vysokohodnotný zážitok pri ich používaní. Ich dostupnosť je vysoká prostredníctvom nainštalovania potrebného *plugina* do webového prehliadača. Najmä zo spomenutých dôvodov sme sa vo vytváranej architektúre obrátili na platformu Silverlight. Pre naše potreby využijeme poslednú verziu, a to Silverlight 4, ktorý podporuje spoluprácu s WCF RIA Services (jeden z hlavných dôvodov výberu týchto dvoch technológií). Popisovaná platforma využíva XAML jazyk pre tvorbu UI. Rozširujúce informácie ohľadom Silverlight 4. je možné nájsť prostredníctvom zdrojov [11], resp. [12].



Obrázok 21: Ukážka Silverlight aplikácie zo systému IDS

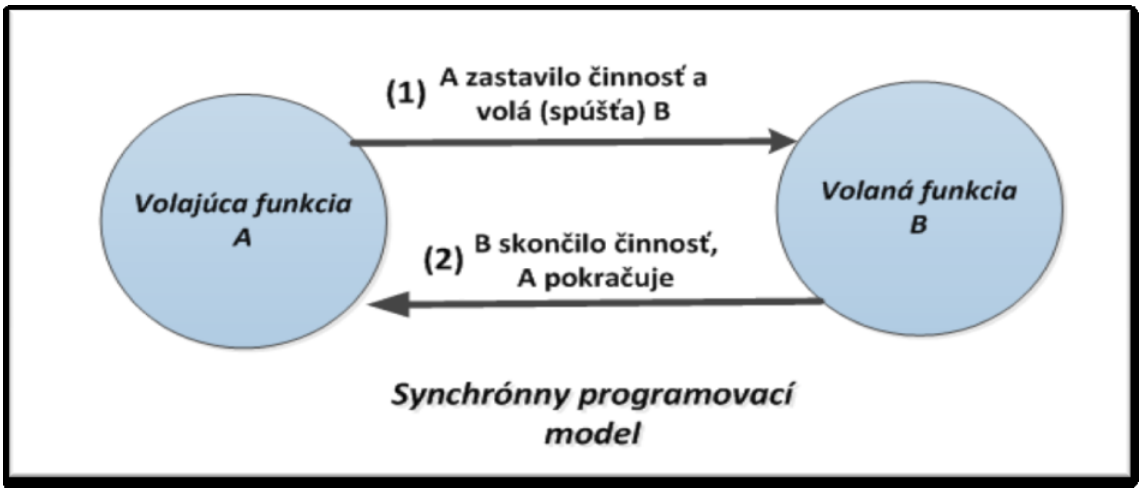
Pri detailnejšom pohľade na zostavenia platformy Silverlight môžeme konštatovať, že triedy, ktoré komunikujú s databázou (napr. ADO.NET) nie sú dostupné. Z tohto dôvodu nie je možná priama komunikácia s databázou. Niet pochybností o tom, že najefektívnejší spôsob, akým Silverlight aplikácia môže získavať dáta z databázy, je prostredníctvom webových služieb. Pomocou spomínaného riešenia je realizovateľná prepojiteľnosť s databázou, resp. možnosť ako nadobudnúť dáta z tretej strany prostredníctvom služieb, a pod. Klientská aplikácia volá metódy požadovanej služby, pričom je úplne zbavená problematiky implementácie na strane serveru.

Ako bolo spomenuté v kapitole 3.2, pre spôsob budovania SOA bol zvolený framework WCF RIA Services. Silverlight 4 a WCF RIA Services sú alternatívy, ktoré do seba veľmi dobre zapadajú a uľahčujú prácu pri vývoji systému. Možnosti, akými tieto dva elementy spolupracujú a vytvárajú požadovanú funkcionalitu, si naznačíme v podkapitolách 5.2.1 a 5.2.2. Základ spolupráce bol načrtnutý v kapitole 3.1.5, kde sme si vysvetlili prepojenie klienta so serverovým projektom pomocou *RIA Linku*. Príklad ako konzumovať webové služby pomocou RIA Services je načrtnutý v prílohe s názvom „Ako tvoriť webové služby pomocou WCF RIA Services".

5.2.1 Komunikácia pomocou asynchrónneho programovacieho modelu

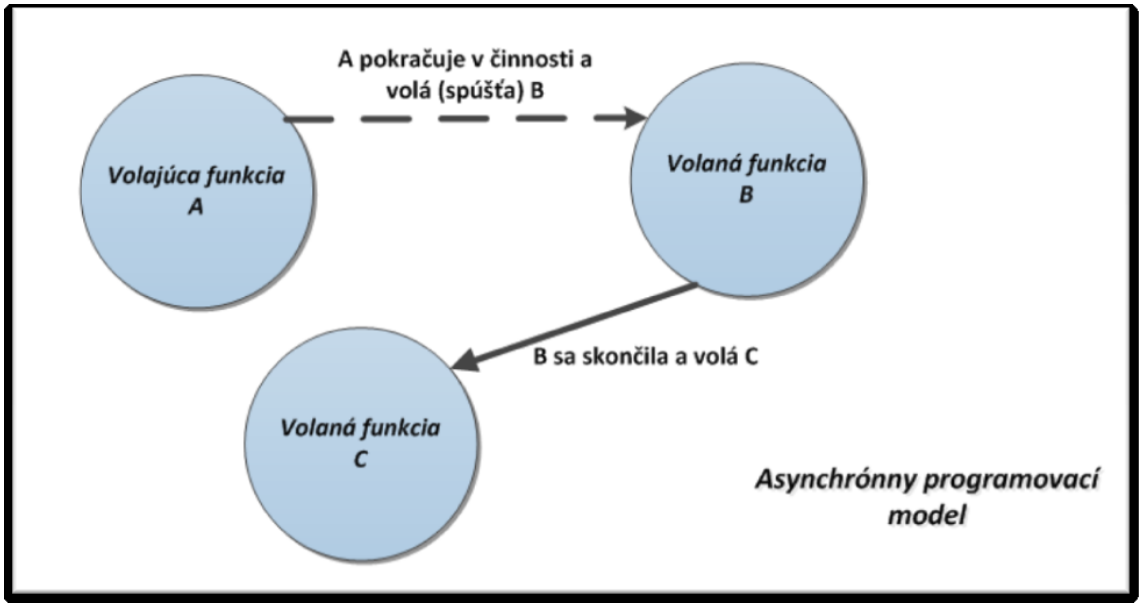
„Zamrznuté" UI neprináša užívateľovi príjemný zážitok práce s aplikáciou, práve naopak. Tento problém môže veľmi ľahko nastať, ak aplikácia, ktorá komunikuje cez Internet používa synchrónny model. Tento model znamená situáciu, kedy *funkcia A* volá *funkciu B*, pričom

funkcia A je v danom momente zastavená. Až po ukončení *funkcie B*, je opäť spustené pokračovanie *funkcie A*. Pre názornú ukážku nám slúži obrázok 22.



Obrázok 22: Princíp synchrónneho programovacieho modelu

Riešením spomenutého problému je asynchrónny programovací model, ktorý je využívaný na platforme Silverlight. Popisovaný model používa dve vlákna. Jedno vlákno sa stará o UI, druhé pracuje na pozadí („*background thread*"). Princíp tejto programovacej techniky spočíva v tom, že *funkcia A* nečaká na ukončenie *funkcie B*, ale pokračuje ďalej, pričom *funkcia B* sa vykonáva paralelne. Keď *funkcia B* skončí, *funkcia A* je oboznámená s touto situáciou a môže na túto udalosť adekvátne reagovať. V Silverlighte je do parametra udalosti (hlásiacej ukončenie *B funkcie*) pridaný výsledok *funkcie B*, v prípade, že nejaký poskytuje. Vďaka asynchrónnym volaniam aplikácia naberá na plynulosti a vylepšuje celkový dojem z nej. Silverlight a WCF RIA Services predstavujú perfektné prostredie pre tvorbu asynchrónneho programovacieho modelu.



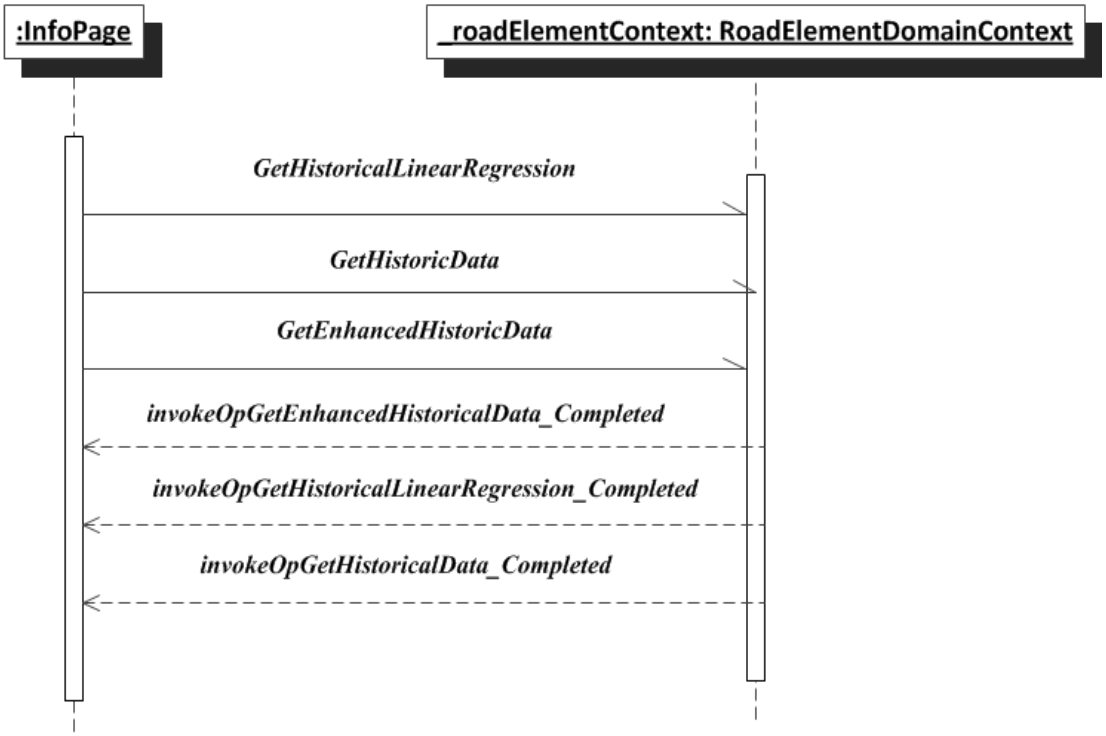
Obrázok 23: Princíp asynchrónneho programovacieho modelu

Využitie asynchrónneho volania je v našej aplikácii veľkou výhodou. Z dôvodu zdĺhavého výpočtu štatistických výstupov nad konkrétnou trasou je vhodné, ako sme už spomenuli na začiatku tejto kapitoly, aby rozhranie aplikácie nezamŕzalo a poskytovalo neustále vysoký stupeň interaktivity s užívateľom. Pre kvalitné objasnenie asynchrónneho volania vo vyvíjanej aplikácii si k výpisu 5 vykreslíme sekvenčný diagram (obrázok 24), kde si priblížime tok činnosti medzi klientom a webovým serverom z časového hľadiska. Užitočné informácie ohľadom popisovanej témy je možné získať na [13].

```
_invOpGetHistLinReg =
    _roadElementContext.GetHistoricalLinearRegression(datetimeFrom,
    predictedDatetime, _selectedLinkID);
_invOpGetHistLinReg.Completed +=
    new EventHandler(invokeOpGetHistoricalLinearRegression_Completed);

_invOpGetHistData =
    _roadElementContext.GetHistoricData(predictedDatetime,
    datetimeFrom, _selectedLinkID);
_invOpGetHistData.Completed +=
    new EventHandler(invokeOpGetHistoricalData_Completed);

_invOpGetEnhHistData =
    _roadElementContext.GetEnhancedHistoricData(predictedDatetime,
    datetimeFrom, _selectedLinkID);
_invOpGetEnhHistData.Completed +=
    new EventHandler(invokeOpGetEnhancedHistoricalData_Completed);
```

Výpis 5: Volanie metód webovej služby z klienta

Z klienta voláme tri metódy smerom na server, menovite *GetHistoricalLinearRegression, GetHistoricData* a *GetEnhancedHistoricData*. Ku každej je nastavená metóda spätného volania, ktorá sa vykoná po skompletizovaní príslušnej metódy na serveri. Ako je vidieť, objekt typu *InfoPage* nečaká na odpoveď objektu *_roadElementContext*, ale pokračuje ďalej v procese.



Obrázok 24: Sekvenčný diagram zobrazujúci komunikáciu medzi klientom a serverom pomocou asynchrónneho programovacieho modelu

5.2.2 Bezpečnosť komunikácie medzi klientom a serverom

Bezpečnosť systému je dôležitý faktor z viacerých pohľadov (neoprávnená manipulácia s dátami neoprávneného užívateľa, a pod.) . V nasledujúcom texte sa pozrieme sa na problémy z hľadiska prístupu klienta do systému. Taktiež popíšeme riešenie a ukážku pre objasnenie problému.

Aby sme zvýšili úroveň bezpečnosti systému, je vhodné publikovať len tie dáta a metódy, ktoré sú na to určené. Ďalším dobrým znakom systému je autentifikácia a autorizácia klienta. Snaha o autentifikáciu a autorizáciu zaisťuje, že sa vykoná len to, čo je povolené z hľadiska manipulácie užívateľa so systémom. Aplikovanie takýchto obmedzení vo WCF RIA Services docielime pomocou atribútu *RequiresAuthenticationAttribute*, ktorý požaduje autentifikáciu užívateľa. Druhým atribútom, ktorým je zvýšená úroveň bezpečnosti, je *RequresRoleAttribute*,

ktorý vyžaduje, aby užívateľ mal určitú rolu. Avšak, naša aplikácia bude zatial využívať iba autentifikáciu, pretože funkcionalita softvéru nenadobudla také rozmery, aby bolo nutné jej obmedzovanie vzhľadom na role užívateľov. Ak v budúcnosti bude potrebné nastavovať oprávnenia užívateľov podľa ich role, tak v kóde stačí použiť spomínaný atribút *RequresRoleAttribute*. Použitie atribútu autentifikácie je naznačené na nasledujúcom výpise 6, kde je atribút použitý na celú službu, z čoho vyplýva, že ak chceme volať hocijakú metódu tejto služby, musíme byť ako užívateľ identifikovaný.

```
[RequiresAuthentication]
public class RoadElementDomainService : DomainService
{
```

Výpis 6: Aplikovanie atribútu *RequiresAuthentcation*

K tomu, aby sme mohli užívateľov identifikovať, je potrebné vytvoriť poskytovateľov, ktorí sa starajú o ich členstvo v systéme. Je nutné každého užívateľa rozpoznať podľa jeho osobných údajov a priradiť mu určitú rolu. Pri riešení tohto problému sa opierame o ASP.NET Membership, avšak za použitia vlastných definovaných poskytovateľov. Týchto poskytovateľov je vhodné budovať na základe nejakej kostry. Touto kostrou sú poskytovatelia ako *MembershipProvider*, resp. *RoleProvider*, ktorý disponujú základnou funkcionalitou. My disponujeme možnosťou vhodne adaptovať ich metódy pre naše potreby. Pre rozšírenie možností užívateľského účtu je prospešné vytvoriť profil pre užívateľa. V tomto smere sa opierame o poskytovateľa profilového charakteru, a to o *ProfileProvider*. Na výpise 7 vytvárame vlastného poskytovateľa *TMembershipProvider*, konkrétne metódu *GetUserInfo*, v spolupráci s Entity Data Modelom.

```
public class TMembershipProvider : MembershipProvider
{

    public UserDB GetUserInfo(string pEmail, bool userIsOnline)
    {
        var userResult = (from user in _studentsEnties.Users
                          where user.email == pEmail
                          select user).FirstOrDefault();

        UserDB userInfo = userResult as UserDB;

        if (userInfo != null)
        {
            userInfo.Security_question = userInfo.Security_questionReference.Value;
            return userInfo;
        }
        return null;
    }
```

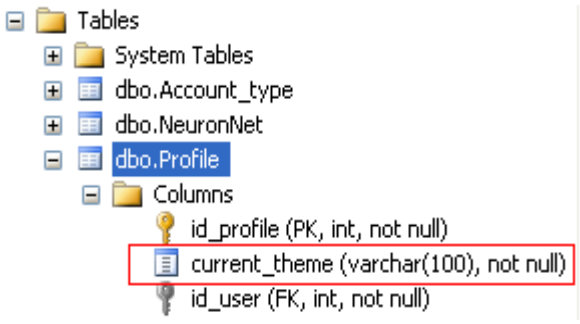
Výpis 7: Tvorba vlastného poskytovateľa *membershipu*

K tomu, aby sme aplikovali nami definovaných poskytovateľov, je potrebné v konfiguračnom súbore pridať kód, zobrazený na výpise 8.

```
<system.web>
<membership defaultProvider="TMembershipProvider">
  <providers>
    <add name="TMembershipProvider"
         type="TrafficSilverlightApplication.Web.TrafficMembership.TMembershipProvider" />
  </providers>
</membership>
<roleManager enabled="true" defaultProvider="TrafficRoleProvider">
  <providers>
    <clear />
    <add name="TrafficRoleProvider"
         type="TrafficSilverlightApplication.Web.TrafficMembership.TRoleProvider" />
  </providers>
</roleManager>
<profile enabled="true" automaticSaveEnabled="false"
         defaultProvider="TrafficProfileProvider">
  <providers>
    <clear />
    <add name="TrafficProfileProvider" tableName="Profile"
         connectionStringName="TrafficMembershipEntities" keyColumnName="id_profile"
         type="TrafficSilverlightApplication.Web.TrafficMembership.TProfileProvider" />
  </providers>
  <properties>
    <add name="CurrentTheme" type="String" customProviderData="current_theme;varchar;100"
         defaultValue="bubble theme" />
  </properties>
</profile>
```

Výpis 8: Nastavenie vlastných poskytovateľov v konfiguračnom súbore

Pre objasnenie predošlého výpisu si popíšeme najdôležitejšie kroky. V elemente *membership* sme pridali nového poskytovateľa autentifikácie, kde sme pomocou atribútu *type* definovali triedu, ktorá tohto poskytovateľa predstavuje. Podobne postupujeme pri zvyšných dvoch poskytovateľoch, *TRoleProvider* a *TProfileProvider*. Pri profile sme navyše špecifikovali tabuľku a pripojovací reťazec k danej DB, taktiež kľúčový atribút tabuľky. Ak chceme obohatiť profil užívateľa o detailnú informáciu (vlastnosť), je potrebné v elemente *properties* pridať danú položku a definovať jej dátový typ a pomocou atribútu *customProviderData* špecifikovať meno atribútu v DB tabuľke a aj jeho dátový typ. Na zobrazenom kóde je pridaná vlastnosť *CurrentTheme* s dátovým typom *String*. Ako môžeme vidieť, popisovaná položka by sa mala vyskytovať v DB tabuľke *Profile* ako atribút *current_theme* s dátovým typom *varchar*.

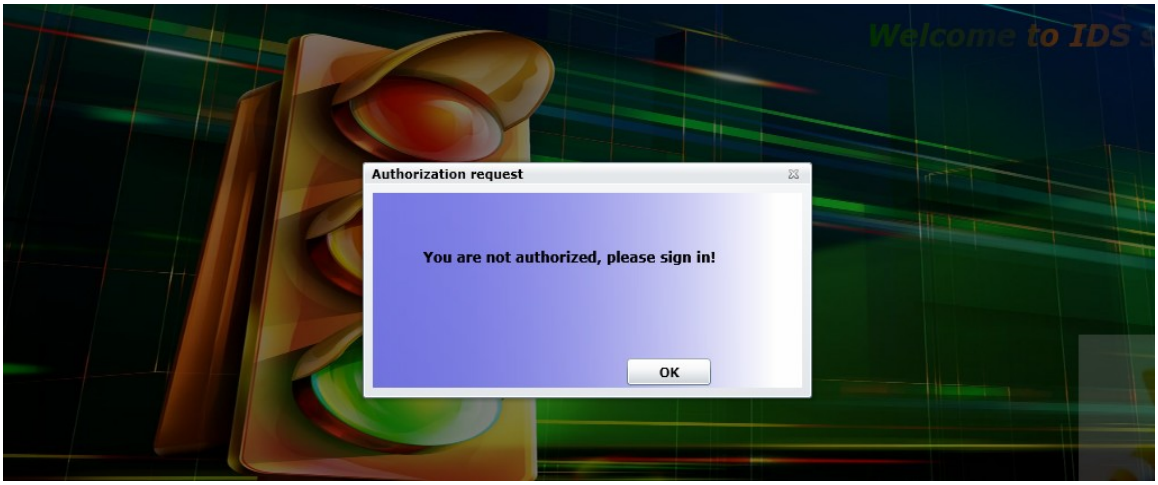


Obrázok 25: Tabuľka *Profile* s definovaným atribútom *current_theme*

Výpis 9 (v programovacom jazyku C#) kontroluje autentifikáciu užívateľa. Ak užívateľ nie je prihlásený, a tak neidentifikovaný systémom, je oboznámený s aktuálnou situáciou a presmerovaný na úvodnú stránku. Ilustrácia na obrázku 26 predstavuje užívateľské rozhranie aplikácie pri popisovanom prípade použitia.

```
if (!WebContext.Current.User.IsAuthenticated)
{
    TAnnounceWindow announceWindow = new TAnnounceWindow();
    announceWindow.AnnounceTextBlock.Text = "You are not authorized, please sign in!";
    announceWindow.Title = "Authorization request";
    announceWindow.OKButton.Click += (s, ee) =>
        {
            Frame parentFrame = this.Parent as Frame;

            if (parentFrame != null)
            {
                parentFrame.Navigate(new Uri("/View/WelcomePage.xaml", UriKind.Relative));
            }
        };
    announceWindow.Show();
```

Výpis 9: Kontrola autentikácie užívateľa s následnou výzvou na prihlásenie sa



Obrázok 26: Užívateľské rozhranie aplikácie oznamujúce neautorizovaný prístup

5.2.3 Klient v spolupráci s mapovou komponentou

Naša aplikácia potrebuje pracovať nad určitým miestom na Zemi. Pre jednoduchú manipuláciu a výber požadovaného bodu je výhodné pracovať s mapovým podkladom. Klient by mal disponovať takouto komponentou pre jednoduchú selekciu, v našom prípade, cestnej komunikácie. Požiadavky na mapový podklad nie sú náročné a mala by ich spĺňať každá komponenta s popisovaným zameraním.

Medzi tieto požiadavky patria:

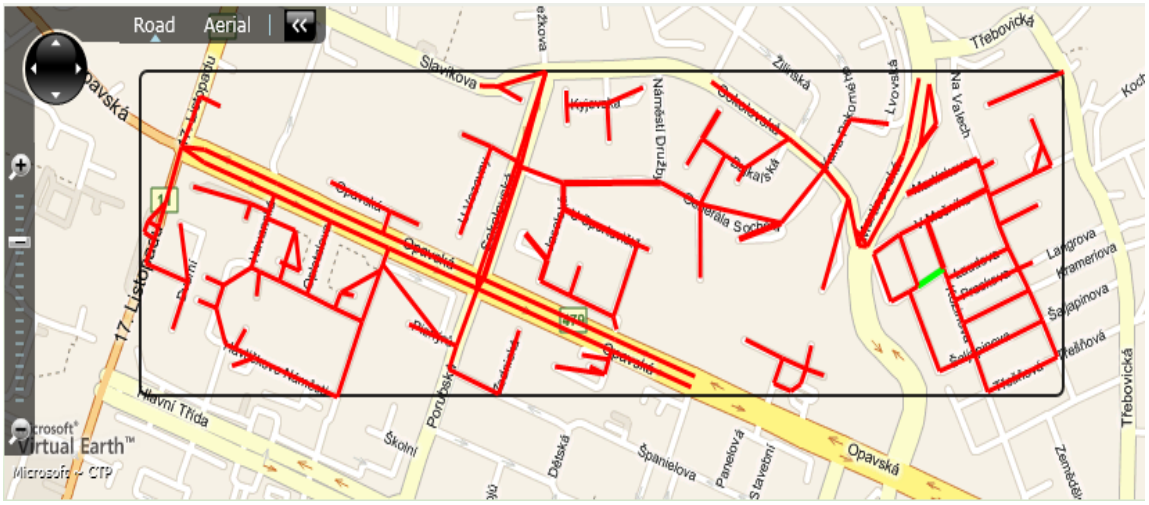
- Kvalitné a prehľadné zobrazenie mapy
- Poskytnutie zemepisných súradníc bodu vybraného myšou
- Možnosť vytvárania vrstiev nad mapovým podkladom

Medzi možné alternatívy, v spojitosti so Silverlight platformou, patria *Silverlight Map Control* od spoločnosti Bing alebo *Telerik Silverlight Map*.

Dôvod výberu *Bing Silverlight Map* komponenty:

- Ľahká dostupnosť
- Spĺňa požiadavky ohľadom funkcionality

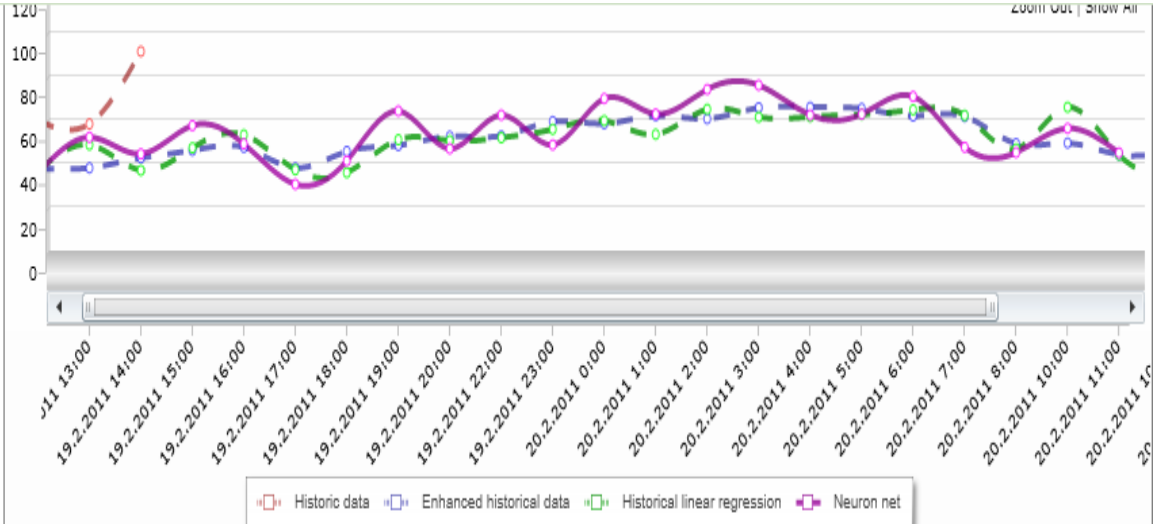
V našej aplikácii máme pomocou popisovanej komponenty nachádzajúcej sa v zostavení *Microsoft.VirtualEarth.MapControl.dll*, možnosť jednoducho vybrať oblasť ciest nášho záujmu (pomocou pravého tlačítka myši). Nad základnou mapovou vrstvou vzniká nová vrstva vyznačujúca všetky komunikácie (sú vyznačené červenou farbou) vo vybranej oblasti. Následne sme oprávnení selektovať cestu (pomocou kliknutia myšou), pre ktorú budú získané štatistické údaje. Pri označení trasy (vyznačená zelenou farbou) je malá tolerancia nepresnosti nad jej výberom, z dôvodu pohodlnejšej manipulácie užívateľa. Príklad kódu vytvárajúceho príkaz nad priestorovými dátami, môžeme nájsť na výpise 4. Podrobné informácie o *Bing Silverlight Map* komponente a o práci s ňou je možné nájsť na [14].



Obrázok 27: Použitá *Bing Silverlight* mapa v aplikácii s vrstvou vyznačujúcou komunikácie

5.2.4 Zobrazenie štatistických výsledkov na klientovi

Prehľadné a pohodlné zobrazenie štatistických výsledkov je jedna zo základných požiadaviek systému. Vďaka takémuto UI by mal užívateľ ľahko pochopiť výsledné hodnoty a transformovať ich do „čitateľnej" podoby. Vytváraná aplikácia ich skonvertuje do podoby krivky v grafe. Pre zobrazenie grafu použijeme komponentu *Chart* zo zostavenia *SLVisifire.Charts.dll*. Spomínané štatistické výsledky nadobúdajú tvar dvojice *rýchlosť-dátum*, a preto v grafe rýchlosť predstavuje x-ovú súradnicu a dátum y-ovú súradnicu. Nad vyselektovanou komunikáciou prebiehajú viaceré výpočty, ktoré odlišujeme rôznymi farbami. Legenda pod grafom vysvetľuje typ každej zobrazenej krivky. Štatistické výsledky majúce charakter popisu komunikácie smerom do minulosti (napr. *Historic Data*) sú vykreslené prerušovanou čiarou. Naopak krivky popisujúce predikčné dáta o trase sú vyznačené plnou čiarou (napr. *Neuron Net*). Dáta obsiahnuté v DB nezobrazujú celkovú realitu na cestách. Pre niektoré trasy neboli poskytnuté informácie. Z tohto dôvodu nie je možné pre tieto komunikácie zobraziť výstupy.



Obrázok 28: Komponenta *Chart* zobrazujúca štatistické výsledky

5.3 Popis ďalších komponent systému

Na obrázku 15 sú zobrazené viaceré, doposiaľ nespomenuté časti systému. Na význam a funkcionalitu niektorých z nich, sa pozrieme v tejto stručnej podkapitole.

Logická vrstva spĺňa funkciu jadra každého biznis procesu. Jej primárnou úlohou je spájanie ostatných modulov systému a taktiež správne nasmerovanie toku biznis procesu požadovaným smerom. Táto komponenta by mala prevažne pracovať s biznis entitami, ktoré sú definované v zostavení *BusinessEntites*.

TrafficBase je zostavenie obsahujúce triedy, ktoré nám určitým spôsobom majú uľahčovať implementáciu. Poskytuje miesto, kde je vhodné vytvárať kód, ktorý nám zefektívni vývoj. Tento kód je všeobecný, a svojím charakterom nezapadá do konkrétnej vrstvy. Jeho funkcionalitu by mali byť schopné využiť všetky vrstvy (logická vrstva, servisná vrstva, a pod.) pre svoje interné účely.

6 Hodnotenie vytváraného systému

Aplikačné jadro systému predstavuje kvalitne pracujúci mechanizmus, ktorý je adaptovateľný do iných systémov. Ide o systém schopný dynamicky pracovať s jeho okolím v distribuovanom prostredí. Jeho architektúra je schopná zabudovať nové moduly, resp. časti, pre rozšírenie funkcionality produktu. Pomocou SOA architektúry vytvárame návrhový vzor *Fasáda*, kde servisná vrstva predstavuje „vstupný bod" klienta do systému. Pri voľbe spôsobu tvorby SOA architektúry, bol primárny faktor využitie Silverlight klienta, preto sme použili WCF RIA Services.

Dobrým znakom programovania je využívanie sofistikovaných prvkov programovania, ktoré nám zjednodušia prácu. Tak ako samotný vývoj systému by sa mal opierať o najnovšie technológie, tak podobne by spôsob písania kódu mal predstavovať určitú úroveň kvality. Túto kvalitu by mal programátor dosiahnuť v spolupráci s používanou platformou. V našom kóde, ako príklad kvalitného a efektívneho kódu v spojitosti s platformou .NET, uvedieme použitie reflexie a generických dátových typov. Príklad je zobrazený na výpise 10. O reflexii je možné získať informácie na [16], resp. o generických dátových typoch na [17].

```
public class ReflectionHelper<O,I> where I:class where O:class,new()
{
    public O ConvertToBE(I pType)
    {
        O outputObject = new O();

        Type type = typeof(I);
            if (type.Name.Equals(typeof(I).Name))
            {
                foreach (PropertyInfo propertyItem in type.GetProperties())
                {
                    foreach (PropertyInfo pItem in outputObject.GetType().GetProperties())
                    {
                        if (propertyItem.Name.ToLower().Equals(pItem.Name.ToLower()))
                        {
                            pItem.SetValue(outputObject, propertyItem.GetValue(pType,null), null);
                            break;
                        }
                    }
                }
            }
        return outputObject;
    }
}
```

Výpis 10: Využitie reflexie a generických dátových typov pri implementácii systému

Keďže pracujeme s RIA Services, je nutné, aby pri nasadení webový server disponoval .NET Frameworkom verzie 4, a tiež RIA Services, ktoré budú poskytovať potrebné zostavenia pre chod aplikácie. Prostredie, do ktorého môžeme aplikáciu umiestniť, je zobrazené na obrázku 9 (v časti „*Activation and hosting*"). Pre správne správanie sa aplikácie, je potrebné vhodne nastaviť konfiguračný súbor *web.config*. Podrobnejšie informácie poskytuje zdroj [18].

Pri reálnom nasadení aplikácie sa zvyčajne stretávame s chybami, ktoré sa pri vývoji nevyskytovali, resp. nepredstavovali vysoký stupeň rizika. Z tohto dôvodu je nutné si uvedomiť, že štatistická analýza, resp. priestorové dáta komunikácií na území celej ČR, pracujú nad veľkým objemom dát. Tento veľký objem spôsobuje dlhšiu dobu čakania na výsledky, pričom čas bude narastať v závislosti na vstupných parametroch. Malé testovanie nad systémom ukázalo, že výkon slabšieho procesora nezvláda požiadavky pre prijateľný chod aplikácie. Ako riešenie je vhodné poukázať na superpočítač, ktorý by rapídne zefektívnil chod a atraktivitu aplikácie.

Dostupnosť produktu dosahuje vysokú úroveň v spojitosti so Silverlight aplikáciou. Ako sme spomenuli, Silverlight je dostupný ako *plugin* do webového prehliadača, ktorý nemusí byť od firmy Microsoft. Avšak je nutné povoliť dva MIME typy, aby bola aplikácia webovým prehliadačom správne identifikovaná. Viac informácií nájdeme prostredníctvom zdroja [19].

7 Záver

Touto písomnou správou som sa podieľal na vzniku systému poskytujúceho štatistické výsledky pre Silverlight aplikáciu prostredníctvom webových služieb. Nosným problémom systému bolo poskytnúť správnu architektúru softvéru. Počas jeho tvorby som sa oboznámil s možnosťami tvorby SOA na platforme .NET pomocou WCF.

Silverlight predstavoval technológiu, ktorá bola pre môj IT svet utajená. Avšak, vďaka zručnostiam získaným, počas tvorby tejto práce, som si objasnil princípy tvorby aplikácie v rámci tejto platformy, čo považujem za jeden z najväčších prínosov. V spojitosti s WCF RIA Services som sa zahĺbil do zjednodušenej techniky vývoja servisne orientovanej architektúry v náväznosti na RIA aplikáciu. Spomenuté technológie mi svojimi možnosťami poskytli vhodné prostredie pre implementáciu požadovaného softvéru. Silverlight 4 v spolupráci s Visual Studio 2010, mi ukázal aj svoju odvrátenú podobu. Vo veľkom množstve prípadov neboli vytvorené informácie umožňujúce sledovanie (*tracing*) a postupné prechádzanie kódu. V takýchto situáciách na klientovi bolo ťažko identifikovať chybu.

Preberaná problematika ma uviedla hlbšie do oblasti priestorových dát. Efektívna manipulácia s týmito dátovými typmi v spolupráci s MS SQL Server 2008, bola pozitívnym bodom pre moje programátorské myslenie.

Samotný vznik aplikácie, ktorý sa odohrával na pozadí písomnej správy, mi ukazoval základné črty procesu pri vytváraní softvérového diela. Iterovaný vývoj softvéru rezultoval do bližšieho kontaktu so zainteresovanou stranou. Spolupráca v tíme znamenala, pre mňa vyšší stupeň zodpovednosti a schopnosť hľadať ideálne riešenie pre kvalitu produktu. Táto spolupráca bola uľahčená verzovacím systémom SVN, ktorý predstavoval vhodný spojovací bod medzi vývojármi.

V súčasnosti je systém schopný spolupracovať so štatistickým modulom a poskytovať klientovi výstupy štatistickej analýzy. Štatistický modul, špeciálne vytvorený pre náš softvér, je bližšie popísaný v zdroji [15]. Aplikácia disponuje mapovým podkladom, ktorý poskytuje požadovanú úroveň interaktivity s užívateľom. Je ho možné využiť v inom systéme, ako napríklad Floreon+Traffic. Rozšírenie produktu je možné vďaka navrhnutej architektúre. Systém by mohol byť rozšíriteľný o poskytovanie informácií ohľadom stavu vozovky, havárií, resp. disponovať schopnosťou nájsť najvhodnejšiu trasu z miesta *A* do miesta *B*. Pri takýchto úlohách by sa systém mohol opierať o služby iných systémov, resp. sám vytvoriť modul zaoberajúci sa spomínanou problematikou.

8 Literatúra

[1] RADECKÝ, Michal. Floreon+Traffic : Systém pro modelování, simulace a monitorování dopravy. *IT Systems*[online]. 2009-09-16, 9, [cit. 2009-09-16]. Dostupný z WWW: <http://www.it4i.eu/files/media_itsystems_0909.pdf>

[2] *The Java™ Web Services Tutorial* [online]. Santa Clara (California, USA) : Sun Microsystems, 2006 [cit. 2011-04-26]. Dostupné z WWW: <http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/JavaWSTutorial.pdf>

[3] CHENG, Steven. *Microsoft Windows Communication Foundation 4.0 Cookbook for Developing SOA Applications*. Birmingham (Veľká Británia) : Packt Publishing, 2010. 316 s.

[4] LOWY, Juval. *Programming WCF Services*. 3rd edition. Sebastopol (California) : O´Reilly Media, 2010. 912 s.

[5] SHARP, John. *Windows Communication Foundation 4 Step by Step*. Sebastopol (California) : O´Reilly Media, 2010. 736 s.

[6] CLEEREN, Gill; DOCKX, Kevin. *Microsoft Silverlight 4 Data and Services Cookbook*. Birmingham (Veľká Británia) : Packt Publishing, 2010. 476 s.

[7] LIU, Mike. *WCF 4.0 Multi-tier Services Development with LINQ to Entities*. Birmingham (Veľká Británia) : Packt Publishing, 2010. 348 s.

[8] *Microsoft : MSDN* [online]. 2011 [cit. 2011-04-26]. WCF Data Services. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/cc668792.aspx>.

[9] *Microsoft : MSDN* [online]. 2011 [cit. 2011-04-26]. WCF RIA Services. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ee707344(v=vs.91).aspx>.

[10] Chabroň, Mojmír; Havíček, Petr. Dopravné smerovanie, 2009

[11] ALBERT, Cameron; LA VIGNE, Frank. *Microsoft Silverlight 4 Business Application Development : Beginner's Guide*. Birmingham (Veľká Británia) : Packt Publishing, 2010. 412 s.

[12] MACDONALD, Matthew. *Pro Silverlight 4 in C#*. Spojené štáty Americké : Apress, 2010. 912 s.

[13] TIITINEN, Juha. *Programmer's War Journal : Shortly on Silverlight, WCF and Asynchronous Programming Model*[online]. 2010 [cit. 2011-04-26]. Dostupné z WWW: <http://programmerswarjournal.blogspot.com/2010/06/shortly-on-silverlight-wcf-and.html>.

[14] Microsoft - MSDN [online]. 2011 [cit. 2011-04-26]. Bing Maps Silverlight Control. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ee681884.aspx>.

[15] Horák, Ondrej. Inteligentní dopravní systémy . 2011

[16] *Microsoft - MSDN* [online]. 2011 [cit. 2011-04-26]. Reflection. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/f7ykdhsy.aspx>.

[17] *Microsoft - MSDN* [online]. 2011 [cit. 2011-04-26]. Generics (C# Programming Guide). Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/512aeb7t(VS.80).aspx>.

[18] *Microsoft - MSDN* [online]. 2011 [cit. 2011-04-26]. A Guide to Deploying RIA Services Solutions. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ff426912(VS.91).aspx>.services.

[19] SNEATH, Tim. *MSDN Blogs* [online]. 2008 [cit. 2011-04-26]. Configuring a Web Server to Host Silverlight Content. Dostupné z WWW: <http://blogs.msdn.com/b/tims/archive/2008/03/18/configuring-a-web-server-to-host-silverlight-content.aspx>.

A Manuál pre tvorbu webových služieb

A.1 Ako tvoriť ASMX webové služby

Ak chceme vytvárať webovú služby ASMX, je potrebné najprv vytvoriť webovú aplikáciu, ktorá bude danú službu hosťovať. Pri vytvorení webovej služby ASMX pomocou šablóny *Web Service*, ktorá je dostupná vo vývojovom prostredí Microsoft Visual Studio 2010, nám vzniká súbor s príponou *.asmx*. Do tohto súboru je automaticky pridaná direktíva *WebService*, ktorá naznačuje ASP.NET, že ide o XML webovú službu. V spomínanej direktíve taktiež definujeme jazyk v ktorom je služba napísaná, a definujeme triedu, ktorá predstavuje samotnú webovú službu. Jednoduchý výpis 11 pod nami nám hovorí, že trieda *TestWebService* predstavuje službu napísanú v jazyku C#.

```
<%@ WebService Language="C#" CodeBehind="TestWebService.asmx.cs"
 Class="ASMXWebApplication.TestWebService" %>
```

Výpis 11: Direktíva *WebService* na pozadí webovej služby ASMX

Trieda predstavujúca službu dedí z triedy *WebService*, ktorá predstavuje základnú triedu pre XML webové služby a nachádza sa v mennom priestore *System.Web.Services*. Metódam, ktoré majú byť volateľné prostredníctvom služby, priradíme atribút *WebMethod*. Naopak, tie metódy, ktoré nie sú obohatené o tento atribút, nie sú dostupné z klientskej aplikácie. Nasledujúci implementačný kód na výpise 12, definuje webovú službu *TestWebService*:

```
public class TestWebService : System.Web.Services.WebService
{

    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }

    [WebMethod]
    public Person GetPerson()
    {
        return new Person() { FirstName = "PersonName", Surname = "PersonSurname" };
    }
}
```

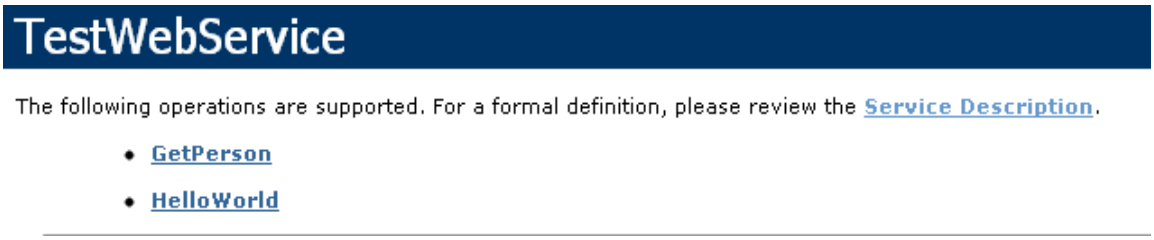
Výpis 12: Definovanie webovej služby ASMX

Dôležité je poznamenať, že ak služba má bežať pod ASP.NET, tak je potrebné pridať do konfiguračného súboru nasledujúci kód, ktorý zabezpečuje kompatibilitu:

```
<system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
</system.serviceModel>
```

Výpis 13: Zabezpečenie kompatibility webovej služby s ASP.NET

Živosť služby je možné overiť kliknutím pravej myši na službu *v solution exploreri* a následným zobrazením vo webovom prehliadači.



Obrázok 29: Overenie behu ASMX služby pomocou webového prehliadača

Vďaka súboru WSDL, ktorý webová služba publikuje, je možné zistiť, akým spôsobom klient má s ňou komunikovať. Ak si chceme zobraziť WSDL súbor, stačí za URL služby pridať reťazec *?wsdl*. Príklad URL a samotného WSDL súboru je nasledovný:

http://localhost:31212/TestWebService.asmx?wsdl

```
- <wsdl:definitions targetNamespace="http://tempuri.org/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      + <s:element name="HelloWorld"></s:element>
      - <s:element name="HelloWorldResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult" type="s:string"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="GetPerson">
          <s:complexType/>
        </s:element>
      - <s:element name="GetPersonResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="GetPersonResult" type="tns:Person"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:complexType name="Person">
        - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="FirstName" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="Surname" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:schema>
    </wsdl:types>
```

Obrázok 30: WSDL súbor webovej služby obsahujúci tiež informácie o dátovom type *Person*

Pomocou Visual Studia resp. pomocou nástroja *SvcUtil.exe* je možné vygenerovať proxy objekt, ktorý sa správa presne ako reálny objekt na serveri. Ak chceme predávať vlastné komplexnejšie dátové typy ako parametre služby, netreba robiť nič nadbytočné. Znova prostredníctvom WSDL je schéma tohto vlastného dátového typu poslaná ku klientovi, takže nám vzniká jej kópia umiestnená na klientskej strane.

A.2 Ako tvoriť webové služby pomocou klasického WCF

Naša úloha v tejto časti je jasná, vyprodukovať službu pomocou WCF. Aby sme ju ľahko zrealizovali, uľahčíme si to znova pomocou vývojového prostredia Visual Studio 2010. V prvom kroku vytvoríme ASP.NET webovú aplikáciu, ktorá bude predstavovať hostiteľské prostredie pre našu WCF službu. Následne pridáme pomocou šablóny *WCF Service* novú službu. Ako výsledok nášho úsilia je pre nás vygenerované rozhranie, ktoré predstavuje kontrakt a služba, ktorá dané rozhranie implementuje. Kontrakt aj triedu si upravíme podľa vlastných potrieb.

```
[ServiceContract]
public interface IProductService
{
    [OperationContract]
    Product GetProduct(int pProductID);

    [OperationContract]
    List<Product> GetProductsUnderUnitPrice(int pUnitPrice);

}
```

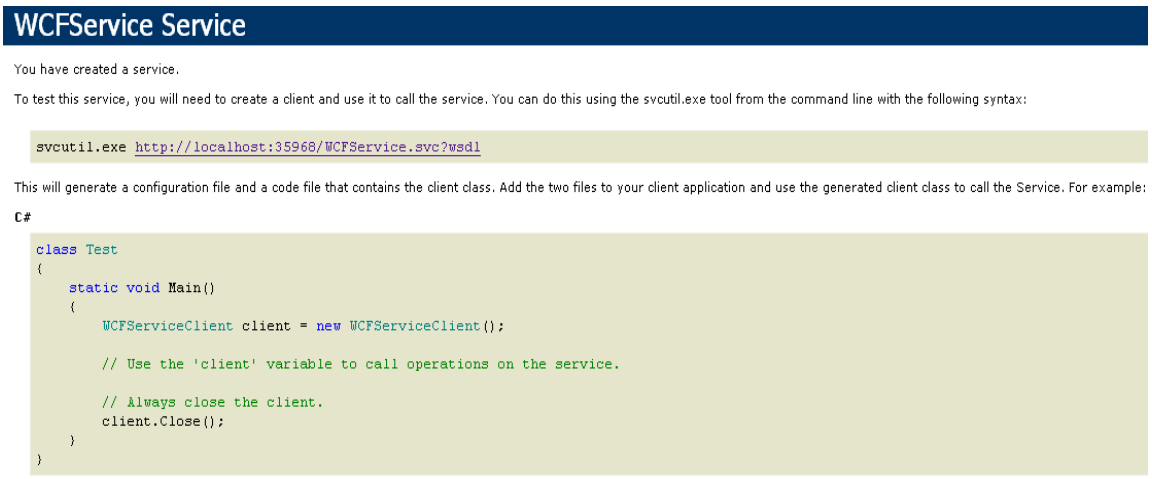
Výpis 14: Definovanie rozhrania webovej služby WCF

Pre definovanie vlastných dátových typov vo WCF využívame atribút *DataContract*. Všetkým jeho vlastnostiam, ktoré majú byť viditeľné pre klientov, priradíme atribút *DataMember*. Príklad dátového kontraktu zobrazuje nasledujúci výpis 15:

```
[DataContract]//namespace System.Runtime.Serialization
public class Product
{

    [DataMember]//namespace System.Runtime.Serialization
    public int ProductID { get; set; }

    [DataMember]
    public string ProductName { get; set; }

    //does not contain DataMember attribute,
    //not included in serialization
    public int UnitPrice { get; set; }

}
```

Výpis 15: Definovanie vlastného dátového kontraktu

Overenie funkčnosti služby vyskúšame kliknutím pravej myši na službu v *solution exploreri* Visual Studia 2010 a následným zobrazením v niektorom webovom prehliadači. Ak vám služba nefunguje, skúste ju otvoriť v inom prehliadači. Správny chod služby vo webovom prehliadači, by mal vyzerať približne takto:



Obrázok 31: Overenie behu WCF služby prostredníctvom webového prehliadača

A.3 Ako tvoriť webové služby pomocou WCF Data Services

Vďaka prostrediu MS Visual Studio 2010 máme dostupnú šablónu *WCF Data Services*. Vytváranú službu môžeme pridať, napríklad do projektu typu *ASP.NET Web Application*. Spomínaná šablóna generuje základnú podobu webovej služby. Vygenerovaná trieda dedí z generickej triedy *DataService*, kde samotný generický typ *T* predstavuje kontajner entít dátového modelu, ktoré chceme publikovať smerom ku klientovi. Tento typ predstavuje *ObjectContext* trieda, ktorá je vygenerovaná z EF modelu (generovanie modelu pomocou EF je popísané v prílohe B.). Na vzorovom výpise 16 pod týmto odstavcom, je touto triedou

CarDatabaseEntities. Je vhodné si najprv vytvoriť model a následne produkovať služby nad ním. Z bezpečnostných dôvodov nie sú implicitne sprístupnené žiadne zdroje z kontajnera. Prístup k jednotlivým entitám kontajnera a manipulovanie s nimi, nastavujeme v metóde *InitializeService* našej služby, a to pomocou metódy *SetEntitySetAccessRule*. Na príklade vidíme, že pre množinu entít *RoadElements*, je povolené len čítanie, kým pri druhej je možné vytvárať nové záznamy.

```
public class RoadDataService : DataService<CarDatabaseEntities>
{
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(DataServiceConfiguration config)
    {

        config.SetEntitySetAccessRule("RoadElements", EntitySetRights.AllRead);
        config.SetEntitySetAccessRule("Roads", EntitySetRights.WriteAppend);
        config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
```

Výpis 16: Nastavenie zdroja webovej služby a nastavenie prístupu k jeho entitám pomocou WCF Data Services

Overiť, či nám naša novovytvorená služba funguje, je možné kliknutím pravej myši na našu službu v *solution exploreri* a jej zobrazením v niektorom webovom prehliadači. Ak všetky kroky boli zrealizované správne, mala by sa nám zobraziť schéma našej služby v Atom formáte.

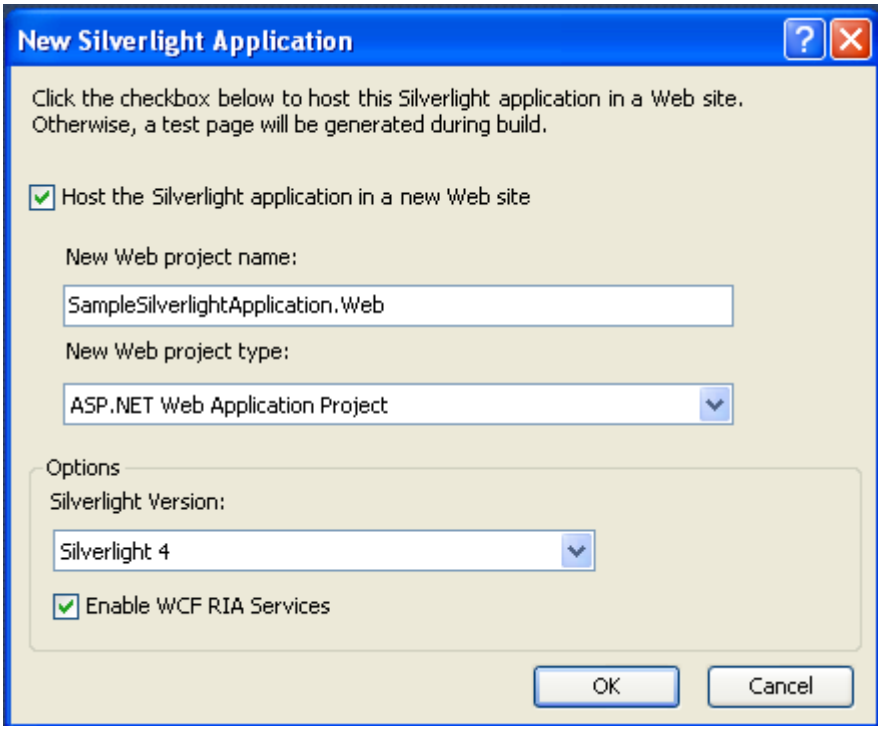
```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://localhost:33709/RoadDataService.svc/" xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns="http://www.w3.org/2007/app">
  - <workspace>
      <atom:title>Default</atom:title>
    - <collection href="RoadElements">
        <atom:title>RoadElements</atom:title>
      </collection>
    - <collection href="Roads">
        <atom:title>Roads</atom:title>
      </collection>
    </workspace>
  </service>
```

Obrázok 32: Atom formát webovej služby

A.4 Ako tvoriť webové služby pomocou WCF RIA Services

Príklad efektívneho využitia WCF RIA Services bude načrtnutý v spolupráci so Silverlight aplikáciou a Entity Framework vo vývojovom prostredí Visual Studio 2010.

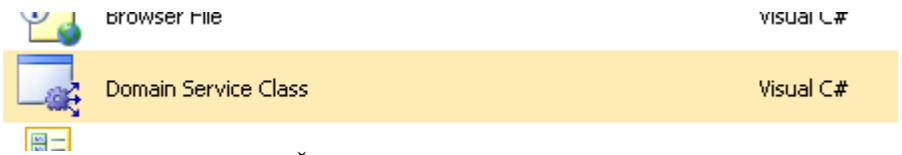
Ako prvý krok vyberieme zo šablón Silverlight aplikáciu. V následnom okne je potrebné vybrať webovú aplikáciu, ktorá bude hosťovať našu aplikáciu. V tomto okne je veľmi dôležité umožniť WCF RIA Services zaškrtnutím checkboxu. Popisovaný framework je možné využiť iba v spolupráci so Silverlight 4. Týmto spôsobom prepojime klienta so serverovým projektom.



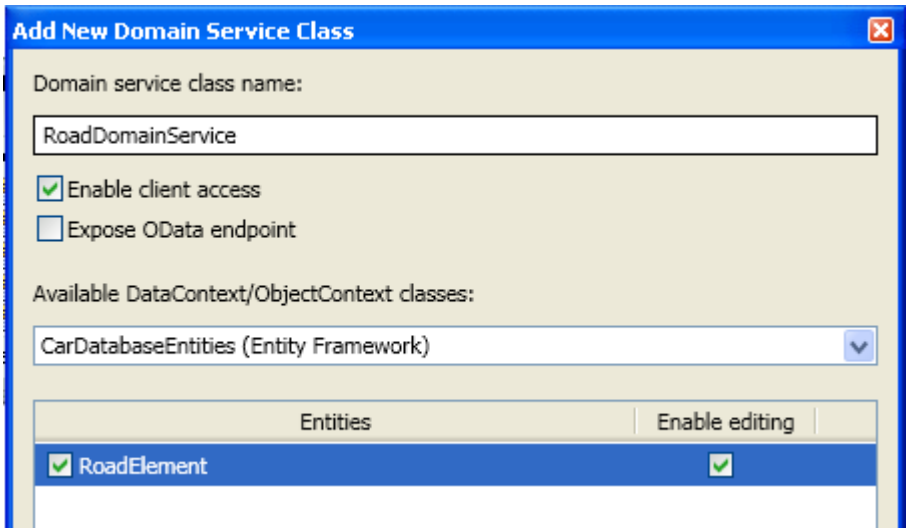
Obrázok 33: Prepojenie Silverlight klienta s webovým projektom a povolenie WCF RIA Services

Keďže jedným z primárnych cieľov budovania SOA služieb je poskytnúť klientom dáta, ďalším krokom je výber dátového zdroja na serveri. Tým môže byť XML súbor, databáza alebo trieda. Pre jednoduchosť znova použijeme Entity Framework a DB, pomocou ktorých vygenerujeme triedu *CarDatabaseEntities*, ktorá dedí z triedy *ObjectContext* a obsahuje vlastnosti, ktoré predstavujú tabuľky v databáze. Následne musíme aplikáciu skompilovať. Spôsob akým generujeme EF model je popísaný v prílohe v časti B.

Dostali sme sa do bodu, kedy je vhodné budovať webové služby. K tomu použijeme šablónu s názvom *Domain Service Class,* zobrazenú na obrázku 34. Ako vyplýva z názvu, doménové služby predstavujú webové služby v kontexte RIA Services. Po potvrdení mena triedy, napríklad *RoadDomainService*, je potrebné vybrať triedu, ktorá predstavuje dátový zdroj pre danú službu. V našom prípade je to vygenerovaná trieda *CarDatabaseEntities*. Nakoniec vyberieme entity, ktoré má služba publikovať. Pre tieto entity môžeme automaticky vytvoriť metódy k ich editovaniu, stačí zaškrtnúť u danej entity checkbox *Enable editing*. Samozrejme, že ľubovolné metódy je možné dopísať explicitne. Popísaný postup je zobrazený na obrázku 35. Nakoniec skompilujeme celú Solution.



Obrázok 34: Šablóna pre WCF RIA Services službu

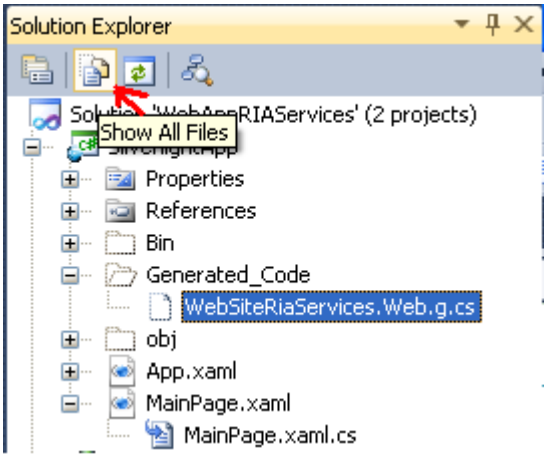


Obr. 35: Nastavenie zdrojovej triedy a konkrétnych entít pre webovú službu pomocou WCF RIA Services

```
[EnableClientAccess()]
public class RoadDomainService : LinqToEntitiesDomainService<CarDatabaseEntities>
{

    public IQueryable<RoadElement> GetRoadElements()...
    public void InsertRoadElement(RoadElement roadElement)...
    public void UpdateRoadElement(RoadElement currentRoadElement)...
    public void DeleteRoadElement(RoadElement roadElement)...
```

Výpis 17: Vytvorená webová služba pomocou WCF RIA Services

Vytvorená trieda je obohatená o atribút *EnableClientAccess*, vďaka ktorému bude rozhranie triedy automaticky vygenerované na strane klienta. Vygenerovaný kód si je možné pozrieť v *solution exploreri*, pozri obrázok 36, kde nad klientským projektom je potrebné kliknúť na tlačítko *Show All Files* a v zložke *Generated_Code* by sa mal nachádzať vygenerovaný kód s príponou *.g.cs*. Keďže sme zvolili dátový zdroj za pomoci EF modelu, bude trieda *RoadDomainService* dediť z generickej triedy *LinqToEntitiesDomainService*.



Obrázok 36: Vygenerovaný súbor na klientovi v *solution exploreri*

Na klientovi je vygenerovaná trieda *RoadDomainContext*, ktorej názov sa odráža od názvu samotnej webovej služby *RoadDomainService*. *RoadDomainContext* automaticky obsahuje metódy, ktoré sa nachádzajú na serveri a slúžia na získanie dát do objektu typu *DomainContext*, ako príklad môže poslúžiť výpis 18. Môžeme si všimnúť, že názvy metód na prezentačnej vrstve sa zhodujú s názvami metód, ktoré ony samé volajú na strane servera, avšak je im navyše priradená koncovka *Query*.

```
public sealed partial class RoadDomainContext : DomainContext
{
    public EntityQuery<RoadElement> GetRoadElementsQuery()
    {
        this.ValidateMethod("GetRoadElementsQuery", null);
        return base.CreateQuery<RoadElement>("GetRoadElements", null, false, true);
    }
```

Výpis 18: Generovaná metóda služby na klientovi pomocou WCF RIA Services

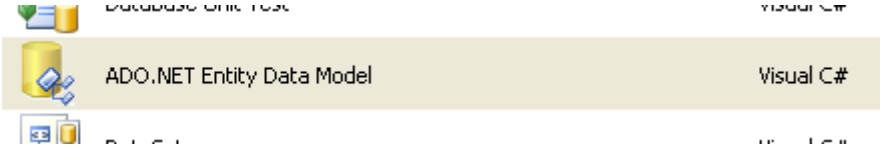
Ako príklad práce s týmito službami v rámci prezentačnej vrstvy, je uvedený výpis 19, ktorý získava všetky entity typu *Road*. Najprv je spustené načítavanie entít. Keďže metódy doménovej služby sú vykonávané asynchrónne, je potrebné spracovať udalosť, keď sa metóda ukončí. Objekt typu *LoadOperation* uchováva výsledok vo svojej vlastnosti *Entities*, cez ktorú pristupujeme k jednotlivým entitám.

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    RoadDomainContext roadCtx = new RoadDomainContext();

    //metoda Completed je volana po ukonceni metody
    roadCtx.Load(roadCtx.GetRoadWithConstraintsQuery(), Completed, null);
}

//metoda roadCtx.GetRoadWithConstraintsQuery() sa ukoncila
void Completed(System.ServiceModel.DomainServices.Client.LoadOperation<Road> loadOp)
{
    if (loadOp.Error == null && !loadOp.CanCancel)
    {
        List<Road> roads = loadOp.Entities as List<Road>;
    }
}
```

Výpis 19: Konzumovanie služby z klienta a spôsob získania výsledkov pomocou WCF RIA Services

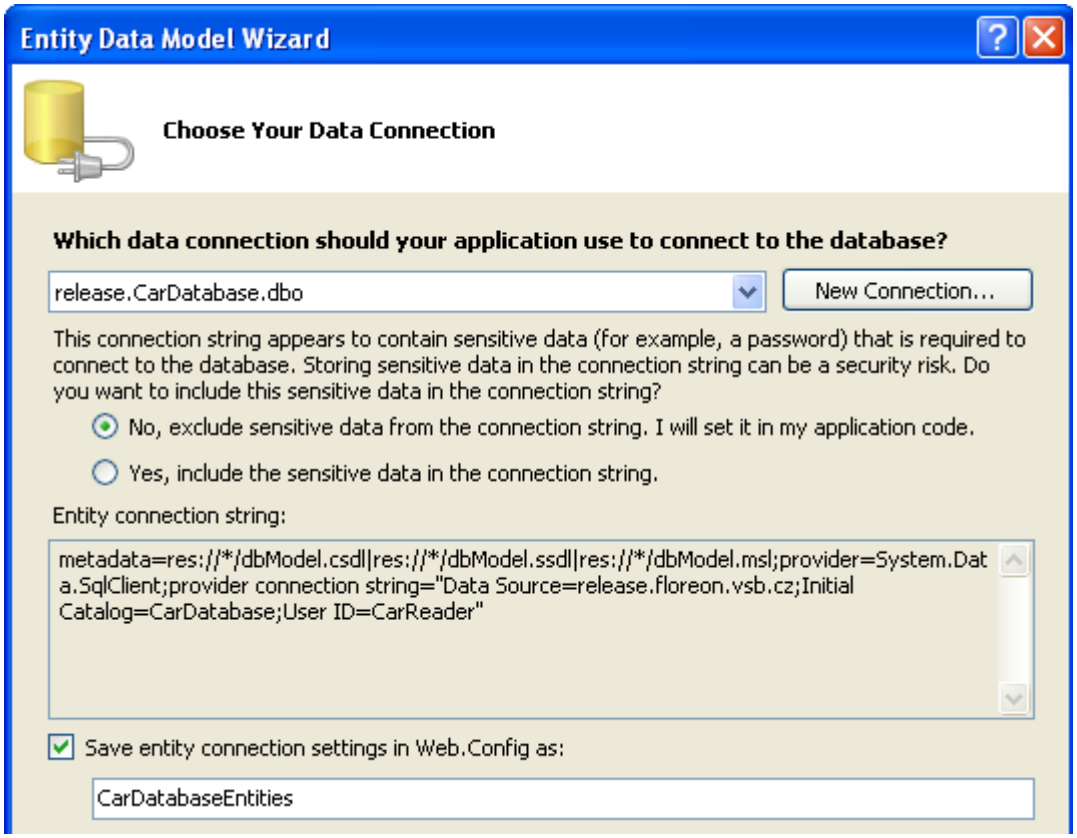
B Tvorba dátového modelu pomocou Entity Framework

Pri zakomponovaní danej technológie do systému využijeme šablónu Visual Studia 2010 s názvom *ADO.NET Entity Data Model* s príponou *.edmx*.

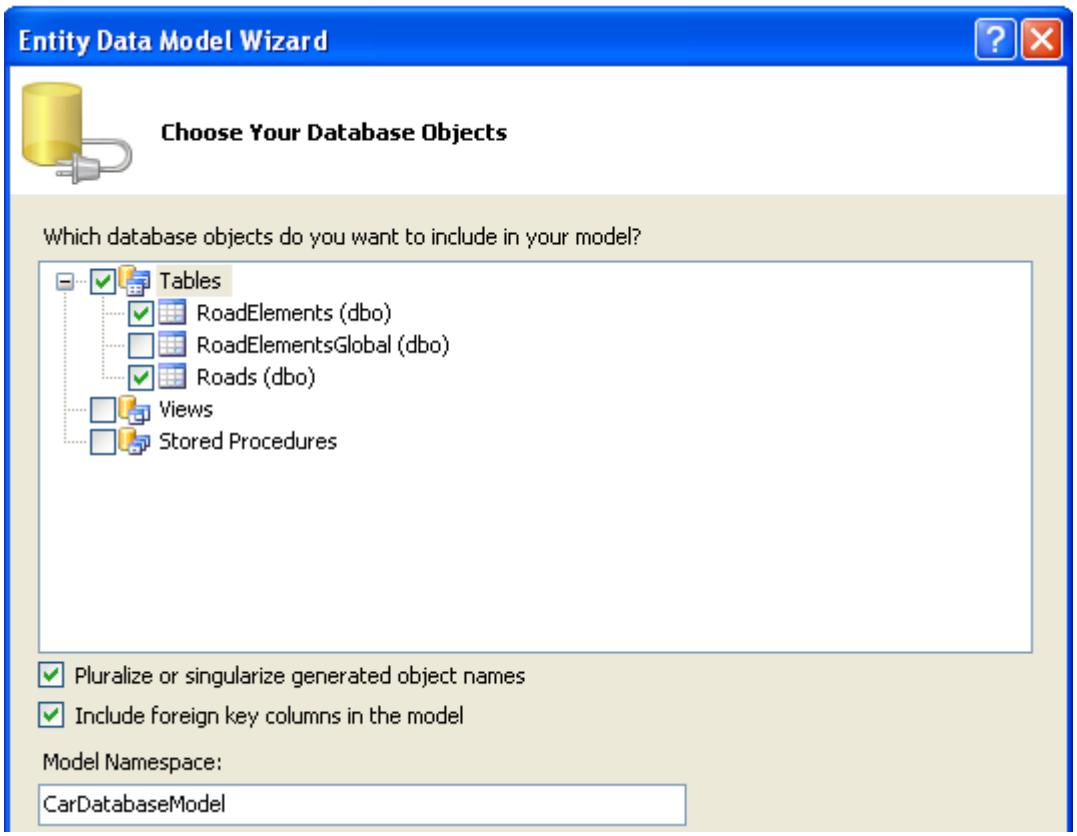


Obrázok 37: Šablóna ADO.NET Entity Data Model

Pomocou sprievodcu zvolíme databázový server a vyberieme objekty z DB, s ktorými chceme v aplikácii pracovať, pozri obrázok 38 a 39. Následne bude vygenerovaný kód, ktorého rozbor už presahuje rámec tejto práce. Na okraj si však môžeme povedať, že je vytvorená trieda, ktorá dedí z rodičovskej triedy *ObjectContext*, ktorá nám predstavuje bránu pre prístup k dátam, ktoré v ďalšej práci využívame.



Obrázok 38: Výber DB serveru obsahujúceho požadovanú databázu



Obrázok 39: Výber objektov z databázy pomocou Entity Data Model sprievodcu